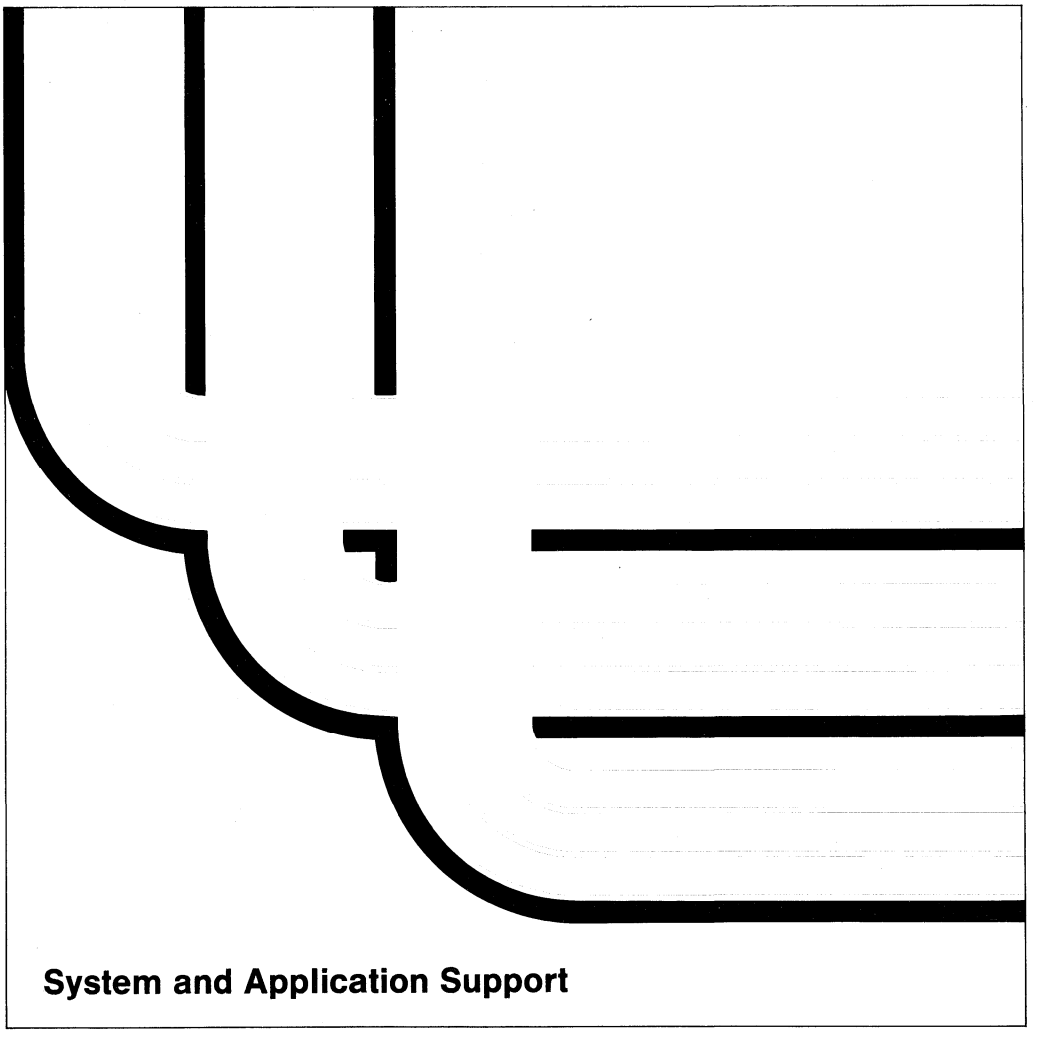
Application System/400

SC41-8223-01

System Programmer's Interface Reference

Version 2



System and Application Support

Application System/400

SC41-8223-01

System Programmer's Interface Reference

Version 2

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page xvii.

Second Edition (September 1992)

This edition applies to the licensed program IBM Operating System/400, (Program 5738-SS1), Version 2 Release 2 Modification 0, and to all subsequent releases and modifications until otherwise indicated in new editions. This major revision makes obsolete SC41-8223-0 and Technical Newsletter SN41-0005-0. Make sure you are using the proper edition for the level of the product.

Order publications through your IBM representative or the IBM branch serving your locality. Publications are not stocked at the address given below.

A Customer Satisfaction Feedback form for readers' comments is provided at the back of this publication. If the form has been removed, you can mail your comments to:

Attn Department 245
IBM Corporation
3605 Highway 52 N
Rochester, MN 55901-7899 USA

or you can fax your comments to:

United States and Canada: 800+937+3430
Other countries: (+1)+507+253+5192

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you or restricting your use of it.

Contents

Part 3. Database File APIs

Part 4. Edit Function APIs

Part 5. Hierarchical File System APIs

Part 6. High-Level Language APIs

Part 7. Message Handling APIs

Part 8. Network Management APIs

Part 9. OfficeVision/400 APIs

Part 10. Operational Assistant APIs

Part 11. Program and CL Command APIs

Part 12. Security APIs

Part 13. Spooled File APIs

Part 14. User Interface APIs

Part 15. User Object APIs

Part 16. Virtual Terminal APIs

Part 17. Work Management APIs

Part 18. Work Station Support APIs

Part 19. Miscellaneous APIs

Part 20. Reference Information

Figures

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

This publication could contain technical inaccuracies or typographical errors.

This publication may refer to products that are announced but not currently available in your country. This publication may also refer to products that have not been announced in your country. IBM makes no commitment to make available any unannounced products referred to herein. The final decision to announce any product is based on IBM's business and technical judgment.

Changes or additions to the text are indicated by a vertical line (|) to the left of the change or addition. Refer to the "Summary of Changes" on page xxi for a summary of changes made to this manual.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This manual contains small programs that are furnished by IBM as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. All programs contained herein are provided to you "AS IS." THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

Programming Interface Information

| This *System Programmer's Interface Reference* is intended to help experienced programmers create application programs.
| This *System Programmer's Interface Reference* documents General-Use Programming Interface and Associated Guidance
| information provided by the Operating System/400 (OS/400) licensed program.

| General-Use programming interfaces allow the customer to write programs that obtain the services of the OS/400
| program.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*), used in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

AD/Cycle	Advanced Function Printing	AFP
Application System/400	AS/400	C/400
COBOL/400	FORTRAN/400	IBM
NetView	OfficeVision	Operating System/2
Operating System/400	Operational Assistant	OS/2
OS/400	Personal System/2	PS/2
RPG/400	SAA	SQL/400
Systems Application Architecture	System/370	400

The following terms, denoted by a double asterisk (**), used in this publication, are trademarks of other companies as follows:

RM/COBOL	Ryan McFarland Corporation
Telex	Telex Computer Products Inc

About This Manual

This manual describes the OS/400 application programming interfaces (APIs). Some APIs provide the same functions as control language (CL) commands. Some APIs provide functions that CL commands do not. Most APIs work more quickly and use less system overhead than the CL commands.

You may need to refer to other IBM manuals for more specific information about a particular topic. The *Publications Guide*, GC41-9678, provides more information on all the manuals in the AS/400 library.

For a list of related publications, see the "Bibliography" on page H-1.

Who Should Use This Manual

This manual is intended for experienced application programmers who are developing system-level and other OS/400 applications. It provides reference information only; it is neither an introduction to the OS/400 licensed program nor a guide to writing OS/400 applications.

Before using the APIs described in this manual, you should be familiar with the concepts discussed in the *Programming: Control Language Programmer's Guide*, SC41-8077. You may need to refer to other IBM manuals for more specific information about a particular topic.

Summary of Changes

This manual uses vertical bars in the left margin to mark technical information that has been added or changed since Version 2 Release 1 of the OS/400 licensed program. Many new APIs have been added, and a few have new functions. The changes include but are not limited to the following:

The documentation for the following APIs has been moved from the *System Programmer's Communications Interface Reference* to this manual.

Figure 0-1. Communications and Virtual Terminal APIs

API	Mnemonic	Location
Close Virtual Terminal Path	QTVCLOVT	Part 16, "Virtual Terminal APIs"
Disable Link	QOLDLINK	Part 2, "Communications APIs"
Enable Link	QOLELINK	Part 2, "Communications APIs"
Open Virtual Terminal Path	QTVOPNVT	Part 16, "Virtual Terminal APIs"
Query Line Description	QOLQLIND	Part 2, "Communications APIs"
Read from Virtual Terminal	QTVRDVT	Part 16, "Virtual Terminal APIs"
Receive Data	QOLRECV	Part 2, "Communications APIs"
Send Data	QOLSEND	Part 2, "Communications APIs"
Send Request for OS/400 Function	QTVSNDRQ	Part 16, "Virtual Terminal APIs"
Set Filter	QOLSETF	Part 2, "Communications APIs"
Set Timer	QOLTIMER	Part 2, "Communications APIs"
Write to Virtual Terminal	QTVWRTVT	Part 16, "Virtual Terminal APIs"

The following APIs were available before Version 2 Release 2 but were not documented.

Figure 0-2. Operational Assistant APIs

API	Mnemonic	Location
Operational Assistant Attention-Key-Handling (group jobs)	QEZMAIN	Part 10, "Operational Assistant APIs"
Operational Assistant Attention-Key-Handling (non-group jobs)	QEZAST	Part 10, "Operational Assistant APIs"
Save Information	QEZSAVIN	Part 10, "Operational Assistant APIs"
Work with Jobs	QEZBCHJB	Part 10, "Operational Assistant APIs"
Work with Messages	QEZMSG	Part 10, "Operational Assistant APIs"
Work with Printer Output	QEZOUTPT	Part 10, "Operational Assistant APIs"

The following are the new APIs for Version 2 Release 2 and the sections of the manual in which they are documented.

Figure 0-3 (Page 1 of 3). New APIs

API	Mnemonic	Location
Add Commitment Resource	QTNADDCR	Part 3, "Database File APIs"
Add List Entry	QUIADDLE	Part 14, "User Interface APIs"
Add List Multiple Entries	QUIADDLM	Part 14, "User Interface APIs"
Add Pop-Up Window	QUIADDPW	Part 14, "User Interface APIs"
Add Print Application	QUIADDPA	Part 14, "User Interface APIs"
Change COBOL Main Program	QLRCHGCM	Part 6, "High-Level Language APIs"
Change Mode Name	QNMCHGMN	Part 8, "Network Management APIs"
Change Object Description	QLICOBJD	Part 15, "User Object APIs"
Change Office Program	QOGCHGOE	Part 9, "OfficeVision/400 APIs"
Change User Password	QSYCHGPW	Part 12, "Security APIs"
Check User Authority to an Object	QSYCUSRA	Part 12, "Security APIs"
Check User Special Authorities	QSYCUSRS	Part 12, "Security APIs"
Close Application	QUICLOA	Part 14, "User Interface APIs"
Close Spooled File	QSPCLOSP	Part 13, "Spooled File APIs"

Figure 0-3 (Page 2 of 3). New APIs

API	Mnemonic	Location
Control Office Services	QOCCTLOF	Part 9, "OfficeVision/400 APIs"
Convert Authority Values to MI Value	QSYCVTA	Part 12, "Security APIs"
Convert Edit Code	QECCVTEC	Part 4, "Edit Function APIs"
Convert Edit Word	QECCVTEW	Part 4, "Edit Function APIs"
Convert Graphic Character Strings	CDRCVRT	Part 19, "Miscellaneous APIs"
Convert Type	QLICVTTP	Part 15, "User Object APIs"
Create Spooled File	QSPCRTSP	Part 13, "Spooled File APIs"
Delete List	QUIDLTL	Part 14, "User Interface APIs"
Deregister Application	QNMDRGAP	Part 8, "Network Management APIs"
Display Directory Panels	QOKDSPDP	Part 9, "OfficeVision/400 APIs"
Display Panel	QUIDSPP	Part 14, "User Interface APIs"
Dump Flight Recorder	QWTDMPFR	Part 17, "Work Management APIs"
Dump Lock Flight Recorder	QWTDMPLF	Part 17, "Work Management APIs"
Edit	QECEDT	Part 4, "Edit Function APIs"
End Application	QNMENDAP	Part 8, "Network Management APIs"
End Data Stream Translation Session	QD0ENDTS	Part 2, "Communications APIs"
Filter Problem	QSXFTRPB	Part 8, "Network Management APIs"
Get CCSID for Normalization	CDRGCCN	Part 19, "Miscellaneous APIs"
Get Dialog Variable	QUIGETV	Part 14, "User Interface APIs"
Get Encoding Scheme	CDRGESP	Part 19, "Miscellaneous APIs" on page 38-3
Get List Entry	QUIGETLE	Part 14, "User Interface APIs"
Get List Multiple Entries	QUIGETLM	Part 14, "User Interface APIs"

Figure 0-3 (Page 2 of 3). New APIs

API	Mnemonic	Location
Get Related Default CCSID	CDRGRDC	Part 19, "Miscellaneous APIs"
Get Short Form CCSID	CDRSCSP	Part 19, "Miscellaneous APIs"
Get Space Status	QLYGETS	Part 6, "High-Level Language APIs"
Get Spooled File Data	QSPGETSP	Part 13, "Spooled File APIs"
List Authorized Users	QSYLAUTU	Part 12, "Security APIs"
List Database Relations	QDBLDBR	Part 3, "Database File APIs"
List Job Schedule Entries	QWCLSCDE	Part 17, "Work Management APIs"
List Objects Secured by Authorization List	QSYLATLO	Part 12, "Security APIs"
List Objects That Adopt Owner Authority	QSYLOBJP	Part 12, "Security APIs"
List Objects User Is Authorized To or Owns	QSYLOBJA	Part 12, "Security APIs"
List Users Authorized to Object	QSYLUSRA	Part 12, "Security APIs"
Open Display Application	QUIOPNDA	Part 14, "User Interface APIs"
Open Print Application	QUIOPNPA	Part 14, "User Interface APIs"
Open Spooled File	QSPOPNSP	Part 13, "Spooled File APIs"
Print Panel	QUIPRTP	Part 14, "User Interface APIs"
Put Dialog Variable	QUIPUTV	Part 14, "User Interface APIs"
Put Spooled File Data	QSPPUTSP	Part 13, "Spooled File APIs"
Query	QQQQRY	Part 3, "Database File APIs"
Read Build Information	QLYRDBI	Part 6, "High-Level Language APIs"
Receive Data	QNMRCVDT	Part 8, "Network Management APIs"
Receive Operation Completion	QNMRCVOC	Part 8, "Network Management APIs"
Register Application	QNMREGAP	Part 8, "Network Management APIs"

Figure 0-3 (Page 3 of 3). New APIs

API	Mnemonic	Location
Remove All Bookmarks from a Course	QEARMVBM	Part 19, "Miscellaneous APIs"
Remove Commitment Resource	QTNRMVCR	Part 3, "Database File APIs"
Remove List Entry	QUIRMVLE	Part 14, "User Interface APIs"
Remove Pop-up Window	QUIRMVPW	Part 14, "User Interface APIs"
Remove Print Application	QUIRMVPA	Part 14, "User Interface APIs"
Retrieve Alert	QALRTVA	Part 8, "Network Management APIs"
Retrieve COBOL Error Handler	QLRRTVCE	Part 6, "High-Level Language APIs"
Retrieve Command Information	QCDRCMDI	Part 11, "Program and CL Command APIs"
Retrieve Commitment Information	QTNRCMTI	Part 3, "Database File APIs"
Retrieve File Description	QDBRTVFD	Part 3, "Database File APIs"
Retrieve Information about a User	QSYRUSRI	Part 12, "Security APIs"
Retrieve Job Description Information	QWDRJOBD	Part 17, "Work Management APIs"
Retrieve Job Queue Information	QSPRJOBQ	Part 17, "Work Management APIs"
Retrieve List Attributes	QUIRTVLA	Part 14, "User Interface APIs"
Retrieve Main Storage	QVTRMSTG	Part 19, "Miscellaneous APIs"
Retrieve Mode Name	QNMRTVMN	Part 8, "Network Management APIs"
Retrieve Office Programs	QOGRTVOE	Part 9, "OfficeVision/400 APIs"
Retrieve Output Queue Information	QSPROUTQ	Part 13, "Spooled File APIs"
Retrieve Program Information	QCLRPGMI	Part 11, "Program and CL Command APIs"
Retrieve Request Message	QMHRTVRQ	Part 7, "Message Handling APIs"
Retrieve User Authority to Object	QSYRUSRA	Part 12, "Security APIs"
Send Error	QNMSNDER	Part 8, "Network Management APIs"

Figure 0-3 (Page 3 of 3). New APIs

API	Mnemonic	Location
Send Message	QEZSNDMG	Part 10, "Operational Assistant APIs"
Send Reply	QNMSNDRP	Part 8, "Network Management APIs"
Send Request	QNMSNDRQ	Part 8, "Network Management APIs"
Set COBOL Error Handler	QLRSETCE	Part 6, "High-Level Language APIs"
Set List Attributes	QUISETLA	Part 14, "User Interface APIs"
Set Lock Flight Recorder	QWTSETLF	Part 17, "Work Management APIs"
Set Screen Image	QUISETSC	Part 14, "User Interface APIs"
Set Space Status	QLYSETS	Part 6, "High-Level Language APIs"
Start Application	QNMSTRAP	Part 8, "Network Management APIs"
Start Data Stream Translation Session	QD0STRTS	Part 2, "Communications APIs"
Translate Data Stream	QD0TRNDS	Part 2, "Communications APIs"
Update List Entry	QUIUPDLE	Part 14, "User Interface APIs"
Write Build Information	QLYWRTBI	Part 6, "High-Level Language APIs"
Work with Problem	QPDWRKPB	Part 8, "Network Management APIs"

The following are the new exit programs for Version 2 Release 2 and the sections of the manual in which they are documented:

Figure 0-4 (Page 1 of 2). New Exit Programs

Exit Program	Location
Action List Option or Pull-Down Field Choice	Part 14, "User Interface APIs"
Application Formatted Data	Part 14, "User Interface APIs"
Commit and Rollback	Part 3, "Database File APIs"
Cursor-Sensitive Prompt	Part 14, "User Interface APIs"
Directory Search	Part 9, "OfficeVision/400 APIs"

Figure 0-4 (Page 2 of 2). New Exit Programs

Exit Program	Location
Directory Verification	Part 9, "OfficeVision/400 APIs"
Document Conversion	Part 9, "OfficeVision/400 APIs"
Document Handling	Part 9, "OfficeVision/400 APIs"
Function Key	Part 14, "User Interface APIs"
General Panel Checking	Part 14, "User Interface APIs"
Incomplete List	Part 14, "User Interface APIs"
Menu Item	Part 14, "User Interface APIs"
Tailoring Automatic Cleanup	Part 10, "Operational Assistant APIs"
Tailoring Operational Assistant Backup	Part 10, "Operational Assistant APIs"
Tailoring the Power Off	Part 10, "Operational Assistant APIs"

Part 1. Introduction to the OS/400 Callable APIs

Chapter 1. Introduction

The OS/400* application programming interfaces (APIs) are for highly experienced programmers who create system applications. For example, programmers can use the APIs to create machine interface (MI) programs and applications for deleting outdated objects.

The APIs are directly callable from high-level language programs. They allow you to:

- Provide better performance when getting system information or when using system functions than is provided by control language (CL) commands.
- Use system information and functions that are not available through CL commands.
- Use calls from high-level languages to these interfaces.

Compatibility with Future Releases

In future releases, IBM intends that one of the following will be true:

- If additional input or output parameters are provided for any of the APIs, the new parameters will be placed after the current parameters. The existing APIs will continue to work without any changes.
- If an additional data structure is provided, a new format (layout of that data structure) will be created.
- New information may be added to the end of an existing format.

To ensure compatibility with future releases, you should retrieve and use all of the following when you work with user spaces generated by list APIs:

- Offset values to the list data section
- Size of the list data section
- Number of list entries
- Size of each entry

OS/400 Terms and Special Values

Before using the OS/400 APIs, you should be familiar with several terms and special values. These terms refer to OS/400 objects:

Program (*PGM). A sequence of instructions that a computer can interpret and run. The system-recognized identifier for the object type is *PGM.

User index (*USRIDX). An object that provides a specific order for byte data according to the value of the data. The system-recognized identifier for the object type is *USRIDX.

User queue (*USRQ). An object consisting of a list of messages that communicate information to other application programs. The system-recognized identifier for the object type is *USRQ.

User space (*USRSPC). An object consisting of a collection of bytes used for storing any user-defined information. The system-recognized identifier for the object type is *USRSPC.

These special values refer to OS/400 libraries and can often be used in API calls in place of specific library names:

***ALL**. All libraries, including QSYS, that the user owns or to which the user has read authority.

***ALLUSR**. All user-defined libraries to which the user has read authority. *ALLUSR *includes* libraries that have names starting with the letter Q and that also contain user data: QDSNX, QGPL, QGPL38, QPRFDATA, QS36F, QUSER38, and QUSRSYS. *ALLUSR *excludes* System/36 libraries that have names starting with the symbol # and that do not contain user data: #CGULIB, #COBLIB, #DFULIB, #DSULIB, #RPGLIB, #SDALIB, and #SEULIB.

***CURLIB**. The job's current library. If no current library is specified for the job, the QGPL library is used.

***LIBL**. The user and system portions of the job's library list.

***USRLIBL**. The user portion of the job's library list.

Summary of OS/400 APIs

Most OS/400 APIs are described in this manual. However, a few special-purpose APIs are described in other books. The sections below outline the APIs described in this manual and list the major operating system APIs described in other manuals.

APIs Described in This Manual

This manual presents the APIs in major functional categories, such as user objects and job management, which correspond to the tabs of this book.

For a complete list of APIs in each part, see the table of contents at the beginning of this manual or the partial table of contents at the beginning of each part. To find a specific API, see the index.

In brief, the major parts of the manual and their contents are as follows:

Part 1, "Introduction to the OS/400 Callable APIs." This part provides an overview of the APIs. It also describes standard API parameters and formats, and discusses considerations in using the APIs in your programs.

Part 2, "Communications APIs." This part provides the information needed to write user-defined communications applications, programming examples, and debugging information. This part also includes the data stream translation APIs, which allow a user-written application program that creates 3270 data streams to run on an Application System/400* (AS/400*) system using 5250 data streams.

Part 3, "Database File APIs." The APIs in this part retrieve information about database files. This part also contains the commitment control APIs, which allow you to add and remove your own resources.

Part 4, "Edit Function APIs." The APIs in this part create and use edit masks.

Part 5, "Hierarchical File System APIs." The APIs in this part let you work with directories and files in hierarchical file systems (HFSs). This section also includes the HFS exit programs.

Part 6, "High-Level Language APIs." The APIs in this part communicate with compilers, and the SQL/400* and COBOL/400* languages.

Part 7, "Message Handling APIs." The APIs in this part allow applications to work with AS/400 messages.

Part 8, "Network Management APIs." The APIs in this part handle alertable messages, work with problem logs, and use the SNA management services transport.

Part 9, "OfficeVision/400 APIs." The APIs in this part work with system distribution directory data and with document handling. This section also includes the OfficeVision/400* exit programs.

Part 10, "Operational Assistant APIs." The APIs in this part provide access to the Operational Assistant* functions.

Part 11, "Program and CL Command APIs." The APIs in this part create programs, retrieve program information, and retrieve command information.

Part 12, "Security APIs." The APIs in this part manage system-level jobs, allowing your programs to run under different user profiles without compromising system security. These APIs also retrieve security information.

Part 13, "Spooled File APIs." The APIs in this part retrieve information and manipulate spooled files.

Part 14, "User Interface APIs." The APIs in this part handle various aspects of the user interface, allowing your applications to display help, display a command line window, convert date and time formats, control keyboard buffering, display screens and pop-up windows, and to build screens. This section also includes the user interface manager exit programs.

Part 15, "User Object APIs." The APIs in this part create, manipulate, and delete user spaces, user indexes, and user queues. They also retrieve information about AS/400* objects.

Part 16, "Virtual Terminal APIs." The APIs in this part provide information needed to use the virtual terminal (VT) APIs, which allow an AS/400 application program to interact with an AS/400 system that is performing work station input/output (I/O).

Part 17, "Work Management APIs." The APIs in this part perform functions that are used in a wide variety of applications. Job APIs retrieve information about jobs. Subsystem management APIs let you retrieve information about subsystems and manipulate subsystem storage pools.

Part 18, "Work Station Support APIs." The APIs in this part allow you to use the type-ahead and attention key buffering functions in your applications.

Part 19, "Miscellaneous APIs." The APIs in this part retrieve information from main storage and remove bookmarks from a course. This part also includes the Character Data Representation Architecture APIs.

Part 20, "Reference Information." This part includes an appendix containing programming examples, a bibliography of related manuals, and an index.

APIs Described in Other Manuals

Several other manuals describe OS/400 APIs that have special uses:

- The *CL Programmer's Guide* describes APIs with general purposes like command checking, as well as APIs for working with data queues:

QCMDEXC	Execute Command
QCMDCHK	Check Command Syntax
QCLSCAN	Scan for String Pattern
QDCXLATE	Translate Fields
QSNDDTAQ	Send Data Queue
QRCVDTAQ	Receive Data Queue
QMHQRDQD	Retrieve Data Queue Description
QCLRDTAQ	Clear Data Queue

- The REXX manuals describe the REXX APIs. See the *Procedures Language (REXX) Programmer's Guide* for information about the following API:

QREXQ	REXX Queue Service

  See the *Procedures Language (REXX) Reference* for information about the following REXX APIs:

QREXVAR	REXX Variable Pool Interface
QREXX	Start REXX Language Processor

- The GDDM manuals describe the GDDM APIs. See the *GDDM Programming Guide* for information about the GDDM API.

Chapter 2. Programming Tips for Using OS/400 APIs

This chapter explains how to use the APIs included in this manual. The chapter covers these topics:

- Language selection
- Data types and parameter coding
- Manipulating user spaces with and without pointers
- User space format for list APIs
- API error reporting
- Performance considerations
- User index considerations (recovering data)
- Partial list considerations

Language Selection Considerations

You can use these APIs with all the languages available on the AS/400* system. Figure 2-1 shows the languages available on the AS/400 system and the data types they provide. For more information, see the manual for the specific programming language you plan to use.

Figure 2-1. Language Selection Considerations

Language[1]	Pointers	Binary 2	Binary 4	Character	Zoned Decimal	Packed Decimal	Floating Point	Structures	Single Array	Exception Handling
BASIC		X	X	X	X		X		X	X
C/400*	X	X	X	X	X[2]	X[2]	X	X	X	X
CL				X		X		X[3]	X[3]	X
COBOL/400*	X	X	X	X	X	X		X	X	X[5]
FORTRAN/400*		X	X	X			X	X	X	
System C/400 PRPQ	X	X	X	X	X[4]	X[4]	X	X	X	X[6]
MI	X	X	X	X	X	X	X	X	X	X
Pascal	X	X	X	X	X[2]	X[2]	X	X	X	X
PL/I	X	X	X	X		X	X	X	X	X
REXX				X				X[3]	X[3]	X
RM/COBOL**		X	X	X	X	X		X	X	
RPG		X	X	X	X	X		X	X	X[6]

Notes:

1. You cannot create Cross System Product (CSP) programs on the AS/400 system. You can create CSP programs on a System/370* and run them on your AS/400 system.
2. There is no direct support, but you can use extended program model (EPM) conversion routines to convert to and from zoned and packed decimal.
3. There is no direct support, but you can use the substring capability to simulate structures and arrays.
4. There is no direct support, but you can use machine interface (MI) instructions to convert to and from zoned and packed decimal.
5. COBOL/400 programs cannot monitor for specific messages, but these programs can define an error handler to run when a program might end because of an error.
6. System C/400 PRPQ and RPG programs cannot monitor for specific messages, but these programs do turn on an error indicator when a called program ends with an error.

Data Types and Parameter Coding

The following sections describe a variety of API coding considerations, covering these topics:

- Data structures
- Character and binary data
- Input and output parameters
- Object name parameters
- Offset values and lengths
- Passing parameters
- Optional parameters

Data Structures

Data structures are available in the QUSRTOOL library for the C/400, System C/400 PRPQ, COBOL/400, and RPG/400* programs.

If you are programming in the RPG language, use the source entry utility (SEU) copy function instead of the RPG /COPY statement when using these data structures. The data structures could change in a future release and recompiling your program could fail if the /COPY statement is used and the structures have changed.

The following table shows where the data structures reside in the QUSRTOOL library:

Language	Source Physical File	Member Name
COBOL/400	QATTCBL	OPxxxxxxxx[1]
C/400 or System C/400 PRPQ	QATTSYSC	OPxxAPI[2] OPxxEXT
General header for a user space[3]	QATTSYSC QATTRPG QATCBL	OPGENHDR OPGENHDR OPGENHDR
RPG/400	QATTRPG	PSxxxxxxxx[1]

Notes:

1 xxxxxxxx is the format name for the API.
2 xx is the component ID (the second and third character of the API name.
3 A diagram showing a general header for a user space is found in "General Data Structure" on page 2-7.

Character Data

In the API parameter tables in this book, CHAR(*) represents character data that has:

- A type that is not known, such as character, binary, and so on
- A length that might not be known

Except for extended attribute parameters, you must enter character-type parameters in uppercase with these APIs. With numeric parameters, you must use a valid length. That is, you must enter a valid length as shown in this manual. If the character parameters are shorter than the numeric length specified, data might be overwritten by the API.

Binary Data

In the API parameter tables in this book, BINARY(2) and BINARY(4) represent numeric data. These parameters must be signed, 2- or 4-byte numeric values with a precision of 15 (halfword) or 31 (fullword) bits and one high-order bit for the sign. Numeric parameters that must be unsigned 4-byte numeric values are explicitly defined as BINARY(4) UNSIGNED.

Input and Output Parameters

API parameters can be used for input or output. Some parameters contain both input and output fields; these are identified as input/output (I/O) parameters in the API parameter tables.

Input parameters and fields are not changed by the API. They have the same value on the return from the API call as they do before the API call. In contrast, output parameters and fields are changed. Any information that an API caller (either an application program or an interactive entry on the display) places in an output parameter or output field before the call will be lost on the return from the call.

Object Name Parameters

Values of all parameters that identify objects on the system must be in *SNAME (simple name) format,[1] left-justified, uppercase, and with valid special characters. The system uses an object name as is, and it does not change or check the object name before locating the object. This improves the performance of the API. An incorrect name usually results in an `Object not found` error.

Offset Values and Lengths

When you are using an API that generates a list into a user space, you should use the offset values and lengths returned by the API in the generic user space header instead of specifying what the current version of the API returns. This is because:

- The offset values to the different sections of the user space may change in future releases.
- The length of the entries in the list data section of the user space may change in future releases.

As long as your HLL application program uses the offset values and lengths returned in the generic header of the user space, your program will run in future releases of the OS/400 licensed program.

Passing Parameters

When you call an API, the protocol for passing parameters is to pass a space pointer that points to the information being passed. This is the convention used by the control language (CL), RPG, and COBOL compilers. Care must be used in some languages that support pointers (such as Pascal, C/400) to ensure that these conventions are followed. Refer to the appropriate language documentation for instructions. The parameter passing conventions can be used in all programming languages.

Some languages, such as the machine interface, allow you to pass the information itself as a parameter to the program being called. You cannot use that convention with the APIs.

1 The *SNAME format is a character string that must begin with an alphabetic character (A through Z, $, #, or @ followed by no more than 9 alphanumeric characters (A through Z, 0 through 9, $, #, @, or _). Periods (.) are not allowed.

Optional Parameters

Some of the APIs have optional parameters; the optional parameters form a group. You must either include or exclude the entire group. You cannot use just one of these parameters by itself. In addition, you must include all preceding parameters.

You may call the API in two ways: either with the optional parameters or without the optional parameters.

Manipulating a User Space with Pointers

Some languages, such as C/400, System C/400 PRPQ, Pascal, and PL/I, support pointers. Pointers allow you to manipulate information more rapidly from the user space. To use pointers with the OS/400 APIs in this manual, you should understand how to:

- Synchronize between two or more jobs
- Use offset values with pointers
- Update usage data

Synchronizing between Two or More Jobs

If you are using the Change User Space (QUSCHGUS) or Retrieve User Space (QUSRTVUS) API to manipulate user spaces, you do not need to synchronize update and retrieve operations when multiple jobs access the user space. The APIs already do that for you. However, if you are using space pointers to retrieve the information directly from the user space, you should synchronize your application programs to avoid data errors. This ensures that no two users update the space at the same time, which can cause unpredictable results.

Locks are typically used to synchronize two jobs on the system, and you can lock user spaces. To synchronize multiple jobs, you can use one of the following:

- Space locks (LOCKSL and UNLOCKSL MI instructions)
- Object locks (LOCK and UNLOCK MI instructions)
- Allocate Object (ALCOBJ) and Deallocate Object (DLCOBJ) commands

Space location locks are faster than object locks. If you do not synchronize two or more jobs, multiple concurrent updates to the user space or read operations can occur while information is being updated. As a result, the data may not be accurate.

Using Offset Values with Pointers

When using a pointer to manipulate the user space, you must:

1. Get a space pointer to the first byte (offset value of zero) of the user space.
2. Retrieve the offset value of the information you want to use from the user space.
3. Add that offset value to the space pointer value.
4. Use the space pointer value to directly refer to the information in the user space.

See "Example: Changing a User Space with a Pascal Program" on page 2-6 for an example of this procedure.

Updating Usage Data

If you are using the Change User Space (QUSCHGUS) or Retrieve User Space (QUSRTVUS) API to manipulate user spaces, you do not need to update usage data information. If you directly retrieve data using pointers, your application programs should update the usage data information. To do this, use the QUSCHGUS API to update the date last changed and use the QUSRTVUS API to update the date last retrieved. You do not need to do this for each retrieve or change operation to the user space, but you should do this once within each application program to maintain accurate usage data information.

Manipulating a User Space without Pointers

When programming in a language that does not support pointers, you can use the Change User Space (QUSCHGUS) and Retrieve User Space (QUSRTVUS) APIs to manipulate data. However, you must first understand how to use positions and lengths with these APIs.

Position Values

Some APIs return offset values into a user space. To use other APIs, such as the Retrieve User Space (QUSRTVUS) API, you must use position values to locate bytes.

Position values and offset values are different ways to express the same thing. An **offset value** is the relative distance of a byte from the first byte of the user space, which has an offset value of 0. A **position value** is the offset value plus 1.

For examples of HLL programs that use positions, see Appendix A, "Examples."

Lengths

List APIs return the length of the information in the different sections of the user space, as well as the length of the list entries in the user space. You should code your application using the lengths returned instead of specifying the current length returned by the API or the size of a data structure in the data structure files. The amount of information returned for any format may increase in future releases, but the information will be placed at the end of the existing information. In order for your application to function properly, it should retrieve the length of the information returned and add that length to a pointer or to a starting position.

Using Offset Values with the Change and Retrieve User Space APIs

When you use the Change User Space (QUSCHGUS) or Retrieve User Space (QUSRTVUS) API, your application program should first retrieve the offset value for the information you want. You must then add one to the offset value to get the starting position for the information.

Example: How the QUSCHGUS API Changes a User Space

Before and after illustrations show how the QUSCHGUS API changes a user space. The following is a user space before you change it with one of the change examples.

```
5728SS1 R03 M00 900824                    AS/400 DUMP          128747/ERICJ/ERICJS1      03/22/90 11:03:26
DMPSYSOBJ PARAMETERS
     TEMPSPACE                      CONTEXT-QGPL
OBJ-
        *USRSPC
OBJTYPE-
OBJECT TYPE-           SPACE                                        *USRSPC
NAME-        TEMPSPACE                      TYPE-          19   SUBTYPE-          34
LIBRARY-     QGPL                           TYPE-          04   SUBTYPE-          01
CREATION-    03/22/90  11:02:56             SIZE-          00000400
OWNER-       ERICJ                          TYPE-          08   SUBTYPE-          01
ATTRIBUTES-          0800                   ADDRESS-       01841400   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934E3C5 D4D7E2D7   C1C3C540 40404040 40404040 40404040  *       - TEMPSPACE             *
  000020   40404040 40404040 E0000000 00000000   00000200 5C800000 00000000 00000000  *        \          *           *
  000040   00000000 00000000 00020002 6E000400   00000000 00000000 00000000 00000000  *           >                  *
SPACE-
  000000   5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C   5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C  **********************************
       LINES  000020    TO    0001FF  SAME AS ABOVE
.POINTERS-
  NONE
OIR DATA-
.TEXT-
  000000   E4A28599 40A29781 83854086 969940C3   88819587 8540E4A2 859940E2 97818385  *User space for Change User Space*
  000020   40C5A781 94979385                                                          * Example                         *
.SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1          *
  000020   40404040 40404040 404040D9 F0F3D4F0   F0F0F9F0 F0F3F2F2 F1F1F0F2 F5F64040  *          R03M000900322110256  *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040  *                               *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                               *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
  0000A0   00000000 00000000                                                          *                               *
END OF DUMP
                                        ***** E N D  O F  L I S T I N G *****
```

Area that will change after using the Change User Space (QUSCHGUS) API

RS3F001-0

The following is a user space after you change it with one of the change examples.

```
5728SS1 R03 M00 900824                    AS/400 DUMP            128747/ERICJ/ERICJS1       03/22/90 11:03:26
DMPSYSOBJ PARAMETERS
     TEMPSPACE                      CONTEXT-QGPL
OBJ-
        *USRSPC
OBJTYPE-
OBJECT TYPE-          SPACE                                        *USRSPC
NAME-        TEMPSPACE                     TYPE-          19   SUBTYPE-          34
LIBRARY-     QGPL                          TYPE-          04   SUBTYPE-          01
CREATION-    03/22/90  11:02:56            SIZE-          00000400
OWNER-       ERICJ                         TYPE-          08   SUBTYPE-          01
ATTRIBUTES-          0800                  ADDRESS-       01841400   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934E3C5 D4D7E2D7   C1C3C540 40404040 40404040 40404040 *       - TEMPSPACE            *
  000020   40404040 40404040 E0000000 00000000   00000200 5C800000 00000000 00000000 *        \          *          *
  000040   00000000 00000000 00020002 6E000400   00000000 00000000 00000000 00000000 *           >                 *
SPACE-
  000000   C2898740 E2A39989 95874097 81848485   8440A689 A3884082 93819592 A2404040 *Big String padded with blanks   *
  000020   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040 *                               *
        LINES  000040    TO    0000BF  SAME AS ABOVE
  0000C0   40404040 40404040 5C5C5C5C 5C5C5C5C   5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C *       *************************
  0000E0   5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C   5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C *********************************
        LINES  000100    TO    0001FF  SAME AS ABOVE
.POINTERS-
  NONE
OIR DATA-
.TEXT-
  000000   E4A28599 40A29781 83854086 969940C3   88819587 8540E4A2 859940E2 97818385 *User space for Change User Space*
  000020   40C5A781 94979385                                                        * Example                       *
.SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040 *                   1           *
  000020   40404040 40404040 404040D9 F0F3D4F0   F0F0F9F0 F0F3F2F2 F1F1F0F2 F5F64040 *          R03M000900322110256  *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040 *                               *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000 *                               *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000 *                               *
  0000A0   00000000 00000000                                                         *                               *
END OF DUMP
                                        ***** E N D  O F  L I S T I N G *****
```

Area changed after using the Change User Space (QUSCHGUS) API

RS3F000-0

Example: Changing a User Space with a Pascal Program

To change the user area of a user space as shown in the previous example with a call from a Pascal program, specify the following:

```
(*********************************************************************)
(***                                                               ***)
(*** PROGRAM:  ChangeUserSpace                                    ***)
(***                                                               ***)
(*** LANGUAGE: PASCAL                                             ***)
(***                                                               ***)
(*** DESCRIPTION:  CHANGE THE CONTENTS OF A USER SPACE            ***)
(***                                                               ***)
(*** APIs USED:  QUSCHGUS                                         ***)
(***                                                               ***)
(*********************************************************************)
program ChangeUserSpace;

  type
    UserSpaceName    = packed array(.1..20.) of char;
    ChangeCharType   = packed array(.1..200.) of char;

  var
    theUserSpace           : UserSpaceName;
    theStartPos, theLength : Integer;
    theNewValue            : ChangeCharType;
    theForceOption         : char;

  procedure QUSCHGUS (var UserSpace: UserSpaceName;
                      var StartPos: Integer;
                      var LengthOfData: Integer;
                      var NewValue: ChangeCharType;
                      var ForceChanges: Char);
                      nonpascal;

begin
  theUserSpace:='TEMPSPACE QGPL';
  theStartPos:=1;
  theLength:=LENGTH(theNewValue);
  theNewValue:='Big String padded with blanks';
  theForceOption:='1';

  QUSCHGUS(theUserSpace,theStartPos,theLength,
           theNewValue,theForceOption);

end.
```

Example: Changing a User Space with an RPG/400 Program

To change the user area of a user space with a call from an RPG/400* program, specify the following:

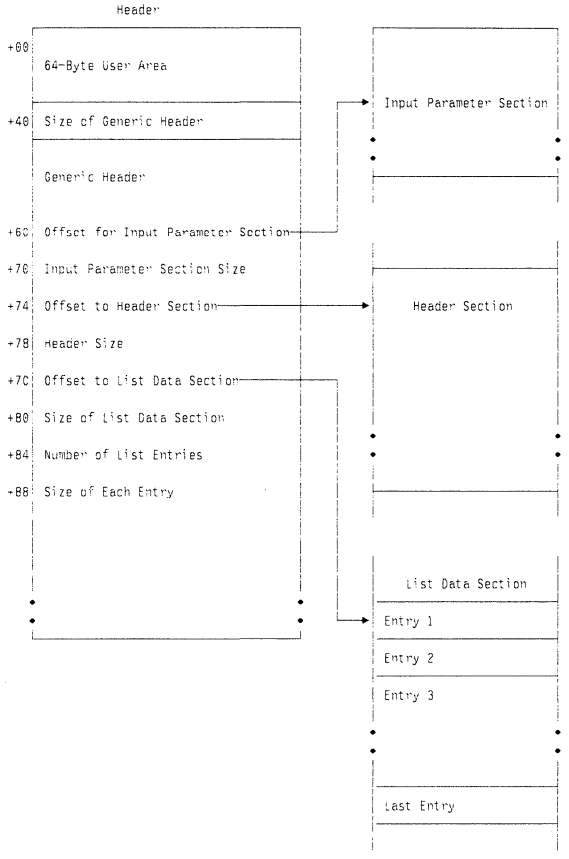
```
H* ****************************************************************
H* ****************************************************************
H*                                                                 *
H*  PROGRAM:  CHANGUS                                              *
H*                                                                 *
H*  LANGUAGE:  RPG                                                 *
H*                                                                 *
H*  DESCRIPTION:  THIS PROGRAM WILL CHANGE THE CONTENTS OF         *
H*                INFORMATION IN THE USER AREA IN THE USER SPACE   *
H*                (FIRST 64 BYTES).                                *
H*                                                                 *
H*  APIs USED:  QUSCHGUS                                           *
H*                                                                 *
H* ****************************************************************
H* ****************************************************************
E                    ARY     1   1 20
E                    CHG     1   1 64
IUSRSPC      DS
I                                        1  10 USNAME
I                                       11  20 USLIB
I            DS
I                                    B   1   40LENDTA
I                                    B   5   80STRPOS
C*                                                                  *
C* ****************************************************************
C* ****************************************************************
C*                                                                 *
C*              OPERABLE CODE STARTS HERE                          *
C*                                                                 *
C* ****************************************************************
C* ****************************************************************
C*                                                                 *
C* MOVE THE USER SPACE AND LIBRARY NAME FROM ARY ARRAY INTO THE    *
C* USRSPC DATA STRUCTURE.  ALSO, MOVE THE NEW USER DATA FROM       *
C* CHG ARRAY INTO NEWVAL.                                          *
C*                                                                 *
C                    MOVELARY,1      USRSPC
C                    MOVELCHG,1      NEWVAL 64
C*                                                                 *
C                    Z-ADD64         LENDTA          LEN OF USERARI
C                    Z-ADD1          STRPOS          STARTING POS
C                    MOVE '1'        FORCE   1       FORCE PARM
C*                                                                  *
C* CALL THE QUSCHGUS API WHICH WILL CHANGE THE USER AREA IN THE    *
C* USER SPACE.                                                     *
C*                                                                 *
C                    CALL 'QUSCHGUS'
C                    PARM            USRSPC
C                    PARM            STRPOS
C                    PARM            LENDTA
C                    PARM            NEWVAL
C                    PARM            FORCE
C*                                                                  *
C* IF MORE USER SPACES NEEDED TO BE CHANGED, THIS PROGRAM COULD    *
C* BE UPDATED TO LOOP UNTIL THE END OF THE ARRAY WAS REACHED.      *
C*                                                                 *
C                    SETON                       LR
C                    RETRN
** ARY
TEMPSPACE QGPL
** CHG
BIG STRING PADDED WITH BLANKS
```

User Space Format for List APIs

To provide a consistent design and use of the user space (*USRSPC) objects, the OS/400 list APIs use a common data structure. The list APIs are those APIs that generate a list unique to that API. This includes any list API that has a user space parameter, such as the List Spooled Files and List Objects APIs.

General Data Structure

The list APIs use the following general data structure:



Note: For more specific information about the format of the generic header and buffer information, refer to "Format of the User Space" on page 26-19.

All offset values are from the beginning of the user space. The offset values for the Dump Object (DMPOBJ) and Dump System Object (DMPSYSOBJ) commands also start at the beginning of the user space. To get the correct starting position for the Change User Space (QUSCHGUS) and Retrieve User Space (QUSRTVUS) APIs, add one to the offset value. The following table shows the generic user space layout. The fields are described in detail after the table.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(64)	User area
64	40	BINARY(4)	Size of generic header
68	44	CHAR(4)	Structure's release and level
72	48	CHAR(8)	Format name
80	50	CHAR(10)	API used
90	5A	CHAR(13)	Date and time created
103	67	CHAR(1)	Information status
104	68	BINARY(4)	Size of user space used
108	6C	BINARY(4)	Offset to input parameter section
112	70	BINARY(4)	Size of input parameter section
116	74	BINARY(4)	Offset to header section
120	78	BINARY(4)	Size of header section
124	7C	BINARY(4)	Offset to list data section
128	80	BINARY(4)	Size of list data section
132	84	BINARY(4)	Number of list data section entries
136	88	BINARY(4)	Size of each entry

Field Descriptions

API used. The name of the API that generated the list.

Date and time created. The date and time when the list was created. The 13 characters are:

1 The century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
2-7 The date, in YYMMDD (year, month, day) format.
8-13 The time of day, in HHMMSS (hours, minutes, seconds) format.

Format name. The name of the format for the input parameter, header, and list data sections.

Information status. Whether or not the information is complete and accurate. Possible values are:

C Complete and accurate.
I Incomplete.
P Partial but accurate.

Number of list data section entries. The number of fixed-length entries in the list data section.

Offset to (all) section. The byte offset from the beginning of the user space to the start of the section.

Size of each entry. The size of each list data section entry, in bytes. All entries are the same size.

Size of generic header. The size of the generic header, in bytes. This does not include the size of the user area;

refer to "General Data Structure" on page 2-7 for a diagram showing the user area.

Size of header section. The size of the header section, in bytes.

Size of input parameter section. The size of the input parameter section, in bytes.

Size of list data section. The size of the list data section, in bytes.

Size of user space used. The combined size of the user area, generic header, input parameter section, header section, and list data section, in bytes. This determines what is changed in the user space.

Structure's release and level. The release and level of the structure. The value of this field is 0100. List APIs put this value into the user space.

User area. An area within the user space that is provided for the caller to use to communicate system programmer-related information between applications that use the user space.

List Sections

Each list API provides the following sections:

List Section	Contents
Input parameter section	An exact copy of the parameters coded in the call to the API. In general, this section contains all the parameters available.
Header section	Parameter feedback and global information about each object. Some APIs do not use this section; in those cases, the value of the size-of-header-section field is zero.
List data section	The generated list data. All entries in the list section are the same length.

When you retrieve list entry information from a user space, you should use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Appendix A, "Examples."

Additional Information about List APIs and a User Space

Before you can use a list API to create a list, the *USRSPC object must exist.

If the user space is too small to contain the list and you have *CHANGE authority to the user space, the list API extends the user space to the nearest page boundary. If the user space is too small and you do not have *CHANGE authority, an authority error results. An extended user space is not truncated when you run the API again.

When you are creating a list into a user space and the user space cannot hold all of the available information (the list is greater than 16 megabytes in length), the API places as much information as possible in the user space and sends a message (CPF3CAA) to the user of the API. The returned list contains only the number of entries that can fit inside the user space (not the total number of entries available).

These APIs retrieve information from internal system objects. Some of the information contains special values. For example, the list object API returns the object type as a special value (*PGM, *LIB, and so on). However, special values may be added in future releases. Even numeric values may have new special values. When you code to these APIs, you should assume that the format of the information returned will not change from release to release, but the content of the information might change.

Example: Listing Database File Members with a CL Program

To generate a list of members that start with **M** and are in file QCLSRC in library QGPL, specify the following:

```
/*****************************************************************/
/*                                                               */
/* PROGRAM:  LSTMBR2                                             */
/*                                                               */
/* LANGUAGE:  CL                                                 */
/*                                                               */
/* DESCRIPTION:  THIS PROGRAM WILL GENERATE A LIST OF MEMBERS,   */
/*               THAT START WITH M, AND PLACE THE LIST INTO A    */
/*               USER SPACE NAMED EXAMPLE IN LIBRARY QGPL.       */
/*                                                               */
/* APIs USED:  QUSCRTUS, QUSLMBR                                 */
/*                                                               */
/*****************************************************************/
PGM
/*****************************************************************/
/*   CREATE A *USRSPC OBJECT TO PUT THE LIST INFORMATION INTO.   */
/*****************************************************************/
  CALL QUSCRTUS                                                  +
       ('EXAMPLE   QGPL      ' /* USER SPACE NAME AND LIB       */ +
        'EXAMPLE   '           /* EXTENDED ATTRIBUTE            */ +
        X'0000012C'            /* SIZE OF USER SPACE            */ +
        ' '                    /* INITIALIZATION VALUE          */ +
        '*CHANGE   '           /* AUTHORITY                     */ +
        'USER SPACE FOR QUSLMBR EXAMPLE                    ')
/*****************************************************************/
/*   LIST THE MEMBERS BEGINNING WITH "M" OF A FILE CALLED        */
/*   QCLSRC FROM LIBRARY QGPL USING THE OUTPUT FORMAT MBRL0200.  */
/*   OVERRIDE PROCESSING SHOULD OCCUR.                           */
/*****************************************************************/
     CALL QUSLMBR                                                +
          ('EXAMPLE   QGPL      ' /* USER SPACE NAME AND LIB    */ +
           'MBRL0200'             /* FORMAT NAME                */ +
           'QCLSRC    QGPL      ' /* DATABASE FILE AND LIBRARY  */ +
           'M*        '           /* MEMBER NAME                */ +
           '1')                   /* OVERRIDE PROCESSING        */
ENDPGM
```

API Error Reporting

The following sections discuss the standard API error code parameter, give examples for its use, and explain how to use the job log to diagnose API errors.

Error Code Parameter

Most OS/400 APIs include an error code parameter to return error codes and exception data to the application. The error code parameter is a variable-length structure containing the information associated with an error condition. One field in that structure is an INPUT field; it controls whether an exception is returned to the application or the error code structure is filled in with the exception information. When the bytes provided field is greater than or equal to 8, the rest of the error code structure is filled in with the OUTPUT exception information associated with the error. When the INPUT field is zero, all other fields are ignored and an exception is returned.

For some APIs, the error code parameter is optional. If you do not code the optional error code parameter, the API returns diagnostic and escape messages. If you do code the optional error code parameter, the API returns only escape messages or error codes; it never returns diagnostic messages.

The structure of the error code parameter is as follows. The fields are described in detail after the table.

Offset		Use	Type	Field
Dec	Hex			
0	0	INPUT	BINARY(4)	Bytes provided
4	4	OUTPUT	BINARY(4)	Bytes available
8	8	OUTPUT	CHAR(7)	Exception ID
15	F	OUTPUT	CHAR(1)	Reserved
16	10	OUTPUT	CHAR(*)	Exception data

Field Descriptions

Bytes available. The length of the exception data available to the API to return, in bytes. If this is 0 (zero), no error was detected.

Bytes provided. The length of the area that the calling application provides for the error code, in bytes. The bytes provided must be 0, 8, or more than 8:

0 If an error occurs, an exception is returned to the application to indicate that the requested function failed.

≥8 and ≤ 32 783
If an error occurs, the space is filled in with the exception information. No exception is returned.

Exception data. A variable-length character field containing the insert data associated with the exception ID.

Exception ID. The identifier for the message for the error condition.

Reserved. A 1-byte reserved field.

Example: Receiving Error Conditions as Exceptions: This example shows an application that receives error conditions as exceptions. It allocates an error code parameter that is a minimum of 4 bytes long. The only field used is the bytes-provided INPUT field, which the application sets to zero to request exceptions. The error code parameter contains the following:

Field	INPUT	OUTPUT
Bytes provided	0	0

Example: Receiving the Error Code without the Exception Data: This application example attempts to create an alert for message ID USR1234 in message file USRMSG in library QGPL. It receives the error condition in the error code parameter but does not receive any insert data. To do this, it allocates an error code parameter that is 16 bytes long—for the bytes provided, bytes available, exception ID, and reserved fields. It sets the bytes-provided field of the error code parameter to 16.

When the application calls the Generate Alert (QALGENA) API, the alert table USRMSG is not found, and QALGENA returns exception CPF7B03. The error code parameter contains the data shown in the following table. In this example, 16 bytes are provided for data, but 36 are available. Twenty more bytes of data could be returned if the bytes-provided field were bigger.

Field	INPUT	OUTPUT
Bytes provided	16	16
Bytes available	Ignored	36
Exception ID	Ignored	CPF7B03
Reserved	Ignored	0

Example: Receiving the Error Code With the Exception Data: This application example attempts to create an alert for message ID USR1234 in message file USRMSG in library QGPL. It receives the error condition in the error code parameter and receives exception insert data as well. To do this, it allocates an error code parameter that is 116 bytes long—16 bytes for the bytes provided, bytes available, exception ID, and reserved fields, and 100 bytes for the insert data for the exception. (In some cases, the insert data might be a variable-length directory or file name, so this might not be large enough to hold all of the data; whatever fits is returned in the error code parameter.) Finally, it sets the bytes-provided field to 116.

When the application calls the Generate Alert API, QALGENA, the alert table USRMSG is not found, and QALGENA returns exception CPF7B03. The error code parameter contains the following:

Field	INPUT	OUTPUT
Bytes provided	116	116

Field	INPUT	OUTPUT
Bytes available	Ignored	36
Exception ID	Ignored	CPF7B03
Reserved	Ignored	0
Exception data	Ignored	USRMSG QGPL

Using the Job Log to Diagnose API Errors

Sometimes an API may issue one or more messages that state that the API failed, and the messages may direct you to see the previously listed messages in the job log. If your application program needs to determine the cause of the error message, you can use the Receive Message (RCVMSG) command to receive the messages that explain the reason for the error. In some cases, you can write an application program to use the diagnostic message to identify and correct the parameter values that caused the error.

Example: Receiving Error Messages from the Job Log: To receive error messages from the job log using a CL program, specify the following:

```
/*                                                                  */
/********************************************************************/
/*                                                                  */
/*    PROGRAM: CLRCVMSG                                             */
/*                                                                  */
/*    LANGUAGE: CL                                                  */
/*                                                                  */
/*    DESCRIPTION: THIS PROGRAM DEMONSTRATES HOW TO RECEIVE         */
/*                 DIAGNOSTIC MESSAGES FROM THE JOB LOG             */
/*                                                                  */
/*    APIs USED: QUSCRTUS                                           */
/*                                                                  */
/********************************************************************/
/*                                                                  */
CLRCVMSG:    PGM

             DCL        VAR(&MSGDATA) TYPE(*CHAR) LEN(80)
             DCL        VAR(&MSGID) TYPE(*CHAR) LEN(7)
             DCL        VAR(&MSGLEN) TYPE(*DEC) LEN(5 0)

             MONMSG     MSGID(CPF3C01) EXEC(GOTO CMDLBL(GETDIAGS))

             CALL       PGM(QUSCRTUS) PARM('!BADNAME  !BADLIB   ' +
                          '!BADEXATTR' -1 '@' '*BADAUTH  ' 'Text +
                          Description')

             /* IF WE MAKE IT HERE, THE SPACE WAS CREATED OK    */

             GOTO       CMDLBL(ALLDONE)

 /* IF THIS PART OF THE PROGRAM RECEIVES CONTROL, A CPF3C01     */
 /* WAS RECEIVED INDICATING THAT THE SPACE WAS NOT CREATED.     */
 /* THERE WILL BE ONE OR MORE DIAGNOSTICS THAT WE WILL RECEIVE  */
 /* TO DETERMINE WHAT WENT WRONG. FOR THIS EXAMPLE WE WILL      */
 /* JUST USE SNDPGMMSG TO SEND THE ID'S OF THE MESSAGES         */
 /* RECEIVED.                                                   */

 GETDIAGS:   RCVMSG     PGMQ(*SAME) MSGQ(*PGMQ) MSGTYPE(*DIAG) +
                          WAIT(3) RMV(*NO) MSGDTA(&MSGDATA) +
                          MSGDTALEN(&MSGLEN) MSGID(&MSGID)
             IF         COND(&MSGID = '       ') THEN(GOTO +
                          CMDLBL(ALLDONE))
             ELSE       CMD(DO)
             SNDPGMMSG  MSG(&MSGID)
             GOTO       CMDLBL(GETDIAGS)
             ENDDO
ALLDONE:     ENDPGM
```

To receive error messages from the job log using an RPG program, specify the following:

```
H* ****************************************************************
H*                                                                 *
H* MODULE: ERRCODE                                                 *
H*                                                                 *
H* LANGUAGE: RPG                                                   *
H*                                                                 *
H* FUNCTION: THIS APPLICATION DEMONSTRATES THE USE OF THE          *
H*           ERROR CODE PARAMETER.                                 *
H*                                                                 *
H* APIS USED: QHFOPNDR, QHFCRTDR                                   *
H*                                                                 *
H* ****************************************************************
H* ****************************************************************
H*                                                                 *
H* THIS PROGRAM DOES SOME SIMPLE VERIFICATION ON AN HFS            *
H* DIRECTORY.  THE QHFRTVAT API IS USED TO VERIFY THE EXISTENCE    *
H* OF THE SPECIFIED DIRECTORY.  IF THE DIRECTORY DOES NOT EXIST,   *
H* AN ATTEMPT IS MADE TO CREATE THE DIRECTORY.                     *
H*                                                                 *
H* THERE ARE THREE PARAMETERS TO THIS PROGRAM                      *
H*                                                                 *
H*     1    INPUT   PATHNM - NAME OF DIRECTORY                     *
H*     2    INPUT   PATHLN - LENGTH OF PATHNM PARAMETER            *
H*     3    OUTPUT  SUCCES - INDICATES SUCCESS OR FAILURE          *
H*                           '0' SUCCESS                           *
H*                           '1' FAILURE                           *
H* ****************************************************************
ISUCCES      DS
I                                       B   1   40RETCOD
IPLENG       DS
I                                       B   1   40PATHLN
IBINS        DS
I                                       B   1   40RETDTA
I                                       B   5   80ATTRLN
IERROR       DS
I                                       B   1   40BYTPRV
I                                       B   5   80BYTAVA
I                                           9  15 ERRID
I                                          16  16 ERR###
I                                          17 256 INSDTA
C           *ENTRY    PLIST
C                     PARM           PATHNM 80
C                     PARM           PLENG
C                     PARM           SUCCES
C*
C* INITIALIZE BYTES PROVIDED AND THE PATHLENGTH VARIABLE
C*
C                     Z-ADD272       BYTPRV
C                     Z-ADD0         ATTRLN
C*
C* OPEN THE DOCUMENT
C*
C                     CALL 'QHFRTVAT'
C                     PARM           PATHNM
C                     PARM           PATHLN
C                     PARM           ATTR    1
C                     PARM           ATTRLN
C                     PARM           ATTR
C                     PARM           ATTRLN
C                     PARM           RETDTA
C                     PARM           ERROR
C*
C* CHECK FOR DIRECTORY NOT FOUND OR FILE NOT FOUND ERRORS.
C* IF WE RECEIVE ONE OF THESE THIS IS THE INDICATION THAT
C* WE CAN TRY TO CREATE THE DIRECTORY.
C*
C           BYTAVA    IFEQ *ZERO
C                     Z-ADD0         RETCOD
C                     ELSE
C           'CPF1F02' IFEQ ERRID
C           'CPF1F22' OREQ ERRID
C* ************************************************************
C* THERE IS NO NEED TO REINITIALIZE THE ERROR CODE PARAMETER.
C* ONLY BYTES PROVIDED IS INPUT TO THE API; IT WILL RESET THE
C* ERROR CODE PARAMETER FOR US.  AFTER THE CALL TO QHFCRTDR,
C* BYTES AVAILABLE WILL EITHER BE 0 IF SUCCESSFUL OR NONZERO
C* IF THE CREATE FAILS.  WE DO NOT HAVE TO WORRY ABOUT THE
C* PREVIOUS ERROR CODE BEING LEFT IN THE ERROR CODE PARAMETER.
C* ************************************************************
C                     CALL 'QHFCRTDR'
```

```
C                   PARM            PATHNM
C                   PARM 20         PATHLN
C                   PARM            ATTR    1
C                   PARM 0          ATTRLN
C                   PARM            ERROR
C           BYTAVA  IFEQ *ZERO
C                   Z-ADD0          RETCOD
C                   ELSE
C                   Z-ADD1          RETCOD
C                   END
C*
C                   ELSE
C                   Z-ADD1          RETCOD
C                   END
C                   END
C*
C* PROGRAM END
C*
C           END     TAG
C                   SETON                       LR
```

Performance Considerations

The retrieve APIs allow you to control the performance cost for information you retrieve. The format specified for any API influences the performance cost of the API. In general, when more information is returned, the performance is slower.

The list APIs, such as list jobs, list spooled files, and list objects, generate the list with minimal cost. This is why these formats do not retrieve very much information. Some of the APIs, such as list record formats and list fields, have only one format, because there is no additional performance cost to supply the complete information.

The retrieve APIs, such as retrieve member description and retrieve spooled file attributes, have formats that are generally ordered from fastest performance to slowest performance. That is, the lower numbered formats run faster but retrieve less information, and the higher numbered formats run slower but retrieve more information. The only exception is the Retrieve Job Information API, QUSRJOBI, where the order of the formats does not have anything to do with performance characteristics. For more information about the performance characteristics for the QUSRJOBI API formats, see "Retrieve Job Information (QUSRJOBI) API" on page 37-15.

User Index Considerations

The performance of a user index is much better than that of a database file. However, before using a user index, you must know the functional differences between a user index and a database file.

The contents of a database file are not affected by an abnormal system end. On the other hand, the contents of a user index may become totally unusable if the system ends abnormally. Therefore, you should not use a user index if the information you want to store needs to remain without errors after an abnormal system end.

If your system abnormally ends when you are removing or inserting a user index entry, unpredictable results may occur. If you are inserting or removing a user index entry and you do not force the index entry to the disk unit using one of the following:

- A user index created with the immediate update parameter set to 1 (affects performance)
- A modify index (MODIDX) MI instruction with the immediate update bit set to 1
- The set access state (SETACST) MI instruction

and the system abnormally ends, your index is probably damaged.

To determine if your last system power down was normal or abnormal, you can check the system value QABNORMSW.

You will not get an error message if your index is damaged. The definition of your index is usable; it is probably the data in your index that is bad.

You can log changes to a database file in a journal, and you can use the journal to apply or remove those changes later. You can also use the journal to audit who is using the database file. However, the system does not support the journaling of indexes. As a result, user applications should log entries in a journal to keep track of changes to the index, but you cannot update the index using apply and remove journal entry functions.

Indexes support the storage of data that does not need to remain after an abnormal system end. If an abnormal system end does occur, you must use a backup copy of the index that was previously saved or create a new copy of the index.

Partial List Considerations

Some APIs may return more information to the application than fits in the user space, which is defined as 20 characters. The information returned is correct, but not complete. If the information is not complete, a P is returned in the information status field of the generic user space layout; refer to "General Data Structure" on page 2-7. If an indicator of a partial list is returned, the application should call the API again with the continuation handle in the list header section of the API and specify that the list begin with the next entry to be returned.

Note: If this is the first time the API is attempting to return information, the continuation handle must be set to blanks.

Part 2. Communications APIs

Chapter 3. Introduction to User-Defined Communications

User-defined communications support is a set of application program interfaces (APIs) that are part of the Operating System/400* (OS/400*) licensed program. These callable routines allow customers to write their own communications protocol stacks above the AS/400* data link and physical layer support. This section of the *System Programmer's Interface Reference* uses the term **user-defined communications** to describe this communications protocol support. The term **application program** refers to a user-defined communications application program.

This section of the manual defines the user-defined communications support, and describes how to write protocols using the APIs. In addition, "User-Defined Communications Programs for File Transfer" on page A-33 provides two C language program examples that illustrate the use of the APIs while performing a simple file transfer between two systems attached to an X.25 packet switched network.

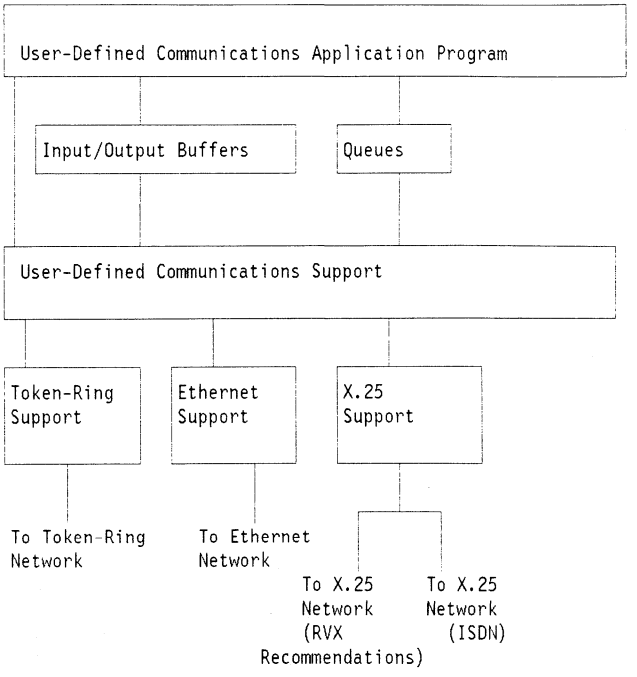
Overview

The user-defined communications APIs allow your application programs to send and receive data, and do specialized functions such as setting timers.

Your application programs need to coexist with the following:

- User-defined communications support
- Input/output buffers and descriptors
- A queue

Figure 3-1 shows an overview of the user-defined communications support.



ISDN means integrated services digital network

Figure 3-1. User-Defined Communications Support

User-Defined Communications Callable Routines

The APIs provided by the OS/400 licensed program are callable routines that allow an application program to start, perform, and end communications, and perform specialized functions such as setting timers. These routines are listed below and are discussed in detail in Chapter 6, "User-Defined Communications Support APIs" on page 6-1.

- Disable Link (QOLDLINK) ends communications
- Enable Link (QOLELINK) starts communications
- Query Line Description (QOLQLIND)
- Receive Data (QOLRECV)
- Send Data (QOLSEND)
- Set Filter (QOLSETF) for inbound routing information
- Set Timer (QOLTIMER) sets or cancels a timer

Input/Output Buffers and Descriptors

The input/output buffers and descriptors are user space objects (*USRSPC) that contain and describe the data an application program is sending or receiving. There are separate buffers and descriptors for input and output.

When an application program is ready to send data, it fills the output buffer with data and provides a description of that data in the output buffer descriptor. Similarly, when an application program receives data, the user-defined communications support fills the input buffer with data and

provides a description of that data in the input buffer descriptor.

The OS/400 licensed program also provides callable APIs to allow an application program to manipulate the data in the user spaces. Some of these APIs are listed below.

- Change User Space (QUSCHGUS)
- Retrieve Pointer to User Space (QUSPTRUS)
- Retrieve User Space (QUSRTVUS)

See Chapter 30, "User Space APIs" on page 30-1 for more information on the user space APIs.

Queues

A queue is used by the user-defined communications support to inform an application program of some action to do or of an activity that is complete.

The OS/400 licensed program provides APIs that allow your application programs to manipulate the data and user queues. Some of these callable APIs are listed below.

- Clear Data Queue (QCLRDTAQ)
- Create User Queue (QUSCRTUQ)
- Delete User Queue (QUSDLTUQ)
- Receive Data Queue (QRCVDTAQ)
- Send Data Queue (QSNDDTAQ)

See the *CL Programmer's Guide* for more information on data queues and the *Languages: System C/400 Programming RPQ P10102 User's Guide and Reference*, SC09-1317, for more information on enqueuing and dequeuing on user queues.

Important Concepts

Listed below are concepts that are important in understanding the information contained in this manual.

Communications handle. The name an application program assigns and uses to refer to a link.

Connection. The logical communication path from one computer system to another. For example, a switched virtual circuit (SVC) connection on an X.25 network.

Connectionless service. A method of operation where data can be sent to and received from the remote computer system without establishing a connection to it. User-defined communications support provides connectionless service over token-ring and Ethernet networks only. For a local area network (LAN) environment, connectionless service is also known as unacknowledged service.

Connection-oriented service. A method of operation where a connection to the remote computer system must first be established before data can be sent to it or received from it. User-defined communications support provides connection-oriented service over X.25 networks only.

Connection identifier. A local identifier (ID) that a computer system uses to distinguish one connection from another. When using the user-defined communications support on the AS/400 system, a connection ID is made up of a user connection end point ID and a provider connection end point ID.

Disable. The process of deactivating a link so that input and output operations are no longer possible on a communications line.

Enable. The process of setting up and activating a link for input and output operations on a communications line.

Filter. The technique used to route inbound data to a link that is enabled by an application program.

Link. The logical path between an application program and a communications line. A link is made up of the following communications objects:

- Network interface description running X.25 over ISDN
- X.25, token-ring, or Ethernet line description
- Network controller description
- Network device description of type *USRDFN

Provider connection end point ID (PCEP ID). The portion of the connection ID that the user-defined communications support uses to identify the connection. For example, data sent by the application program will be on the PCEP ID portion of the connection ID.

User connection end point ID (UCEP ID). The portion of the connection ID that the application program uses to identify the connection. For example, data received by the application program is on the UCEP ID portion of the connection ID.

Relationship to Communications Standards

Figure 3-2 on page 3-3 shows the structure of advanced program-to-program communications (APPC) on the AS/400 system and its relationship to the International Standards Organization (ISO) protocol model. Note that only the application layer above the APPC protocol code is available for definition. The APPC functional equivalents of the ISO presentation, session, networking, transport, data link, and physical layers are performed by OS/400 or Licensed Internal Code, and you can not replace or change them. Contrast this with Figure 3-3 on page 3-3 which shows how much more of the protocol is defined by the user-defined communications application than by the APPC application.

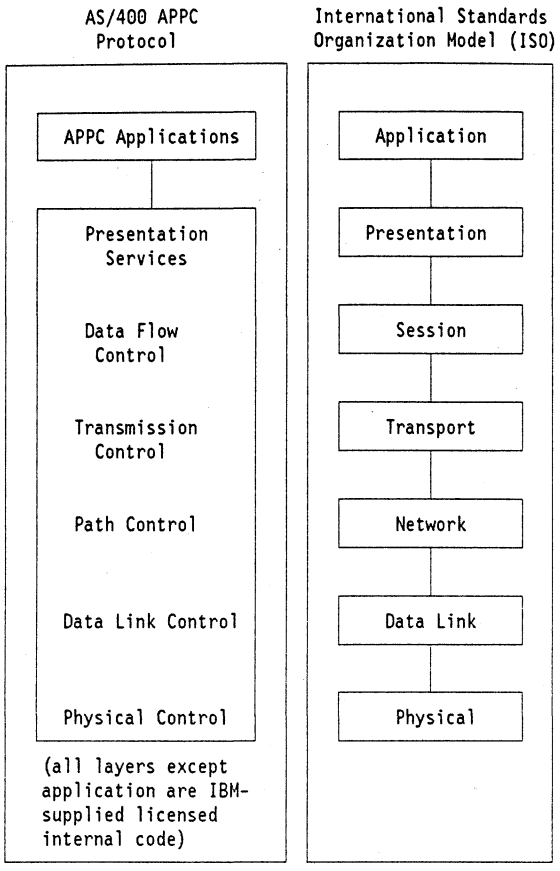


Figure 3-2. AS/400 APPC versus ISO Model

Figure 3-3 shows the new structure for user-defined communications and its relationship to the International Standards Organization (ISO) protocol model. Note that the available AS/400 data links and physical layers limit user-defined communications to run over LAN (token-ring or Ethernet) or X.25 links, but the portion of the protocol above the data link layer is completely open to a user-defined communications application. In addition, these same X.25 and LAN links may be shared between the application program and other AS/400 communications protocols that support X.25 and LAN lines. Examples include Systems Network Architecture (SNA), asynchronous communications, Transmission Control Protocol/Internet Protocol (TCP/IP), and Open Systems Interconnection (OSI).

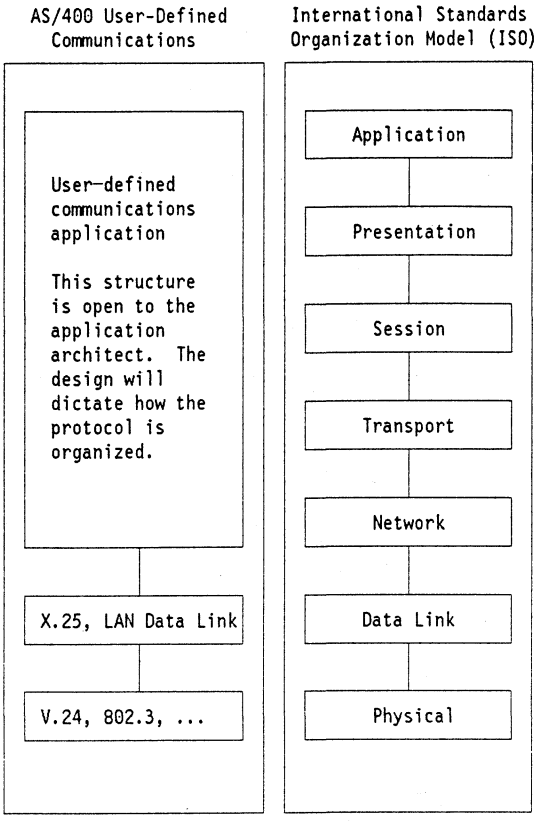


Figure 3-3. AS/400 User-Defined versus ISO Model

You can write protocols that run over local area networks or X.25 networks completely in high-level languages such as C/400*, COBOL/400*, or RPG/400*. You can also write protocols currently running on other systems to run on the AS/400 system. For example, you can write both non-SNA LAN or X.25 packet layer protocols on the AS/400 system.

Configuration instructions also need to be supplied with the application program. User-defined communications support simply opens a pathway to the system data links. It is up to you as a protocol developer to supply any configuration instructions that are in addition to the data link or physical layer definition. Data link and physical layer definitions are defined when you use the following commands:

- Create Line Description (Token-ring) (CRTLINTRN)
- Create Line Description (Ethernet) (CRTLINETH)
- Create Line Description (X.25) (CRTLINX25)
- Create Network Interface Description (ISDN) (CRTNWIISDN)

Figure 3-4 outlines the difference between standard AS/400 communications configuration, such as the AS/400 APPC protocol, and user-defined communications configuration.

Figure 3-4. Comparison between User-Defined Communications and APPC Communications

Object	APPC Communications	User-Defined Communications
Network Interface Description	ISDN basic rate interface (BRI). Describes the physical attachment to an ISDN BRI. Only used for ISDN. X.25 or IDLC protocols supported.	Same as APPC. Only X.25 supported.
Line Description	SDLC, LAN, IDLC, X.25 lines supported. Contains local port information for AS/400 communication IOP (hardware address, maximum frame size, exchange identifier (XID), local recovery information, ...).	LAN, X.25 lines supported. Same as APPC except some of the information does not apply to user-defined communications.
Controller Description	APPC, host controllers supported. Describes remote system, and parameters must match the remote hardware (hardware address, XID, ...).	Network controller supported. Pathway into network. Only one specific parameter—X.25 time-out value.
Device Description	APPC device supported. Describes remote logical unit (LU), and parameters must match partner LU (remote location name, local location name, ...).	Network device supported. Only describes the communications method or type (for example, TCP/IP, OSI, or user-defined communications).
Mode Description and Class-of-Service (COS)	Required.	Not available.

Although an APPC network requires one APPC controller description to describe each remote system in the network, user-defined communications only requires one network controller for communications with an entire network of remote systems. Thus, LAN and X.25 lines can be shared between user-defined communications support and any other protocols that support those same line types. For example, APPC may run over a token-ring line and use the X'04' Service Access Point (SAP). TCP/IP might run at the same time using the X'AA' SAP. You might write an application program to use the X'22' SAP, and run at the same time as the first two. All three protocols can be active at the same time across the same physical media.

Note: System-specific configuration information must be part of the application program and is not supplied by IBM.

Local Area Network (LAN) Considerations

User-defined communications supports three LAN types:

- Token ring (IEEE 802.5)
- Ethernet (IEEE 802.3)
- Ethernet Version 2

For token ring (802.5) and Ethernet (802.3), user-defined communications uses the IEEE 802.2 logical link control (LLC) layer, which provides type 1 connectionless service. Connectionless service is also known as unacknowledged service. The LLC layer provides for type 2 connection service as well. For Ethernet Version 2, no 802.2 layer is available.

Your application program has access to type 1 unnumbered information (UI) frames. This connectionless service is commonly referred to as *datagram* support where protocol data units are exchanged between end points without establishing a data link connection first.

The type 1 operations, test and exchange identifier (XID) frames, are not supported in user-defined communications. Any XID or test frames that the physical layer of the AS/400 system receives are processed by the input/output processor (IOP) and never reach your application program.

LAN frames are routed by filtering incoming data using the inbound routing data defined by your application program. The filters are hierarchical and are set up by your application program before communications is started.

The following list shows the possible settings for LAN inbound routing data (filters) from least selective to most selective.

- Destination Service Access Point (DSAP)
- DSAP, Source Service Access Point (SSAP), and optional Ethernet Version 2 frame type
- DSAP, SSAP, optional Ethernet Version 2 frame type, and adapter address

Because user-defined communications does not allow applications to define the data link and physical layers, the entire token-ring or Ethernet frame is not available to your applications. The following fields are the parts of the LAN frame that are available to the user-defined communications support:

- DSAP
- SSAP
- Destination address (DA)
- Routing information (RI)[1]
- Priority control[1]
- Access control[1]
- Data

For more information on local area networks, see the *Local Area Network Guide*.

X.25 Considerations

X.25 user-defined communications support includes access to both permanent virtual circuits (PVCs) and switched virtual circuits (SVCs).

Over X.25 networks, including those using ISDN, your application program can initiate and accept X.25 calls, send and receive data, reset, and clear connections.

X.25 packets are routed by filtering the incoming call request using the inbound routing data that is defined by your application program. The filters are hierarchical and are set up by the application program before communications is started.

The following list shows the possible settings for X.25 inbound routing data (filters) from least selective to most selective.

- Protocol identifier (PID)
- PID, and calling data terminal equipment (DTE) address

When X.25 networks are using ISDN, notification of incoming calls may be received on the D-channel. You can decide whether these calls are accepted.

For more information on X.25 networks, see the *X.25 Network Guide*.

[1] These fields are available only when using token ring.

Chapter 4. Programming Design Considerations

This chapter discusses concepts related to user-defined communications and how they might relate to the design of a user-defined communications application. This chapter covers:

- Jobs
- Application program feedback
- Programming languages
- Connection identifiers
- Token-ring and Ethernet networks
- X.25 networks
- Queues
- User spaces

Jobs

A fundamental concept in user-defined communications is the job. The concept of the job is important because the user-defined communications support performs services for the job requesting the communications support through one of the user-defined communications APIs. Information used by the user-defined communications support is kept along with other information about the job. You can display this information by using the Work with Job (WRKJOB) command and selecting the Work with communications status option. The user-defined communications information for the job, such as the communications handle name, last operation, and input and output counts are shown.

A user-defined communications application program (hereafter referred to as an application or application program), always runs within a job. This job may be run interactively or in batch and always represents a separate application to the user-defined communications support. This means that the same protocol can be actively running in more than one job on the system. Also, more than one job can have links that share the same line as other jobs running application programs.

Each link that is enabled by an application program logically consists of the line, network controller, and network device description objects (plus the network interface description object for ISDN links). Many applications can share the same line and controller description, provided the applications are running in different jobs, but each application uses a different device description. Up to 256 device descriptions can be attached to a controller description. This means that there can be a maximum of 256 jobs running application programs that share the same line at one time. When an application program has finished using a link and disabling it, the network device description used by the application becomes available to another application.

For end-to-end communication to begin, the application programs on each system must be started. There is no function equivalent to the intersystem communications function (ICF) program start request. Your application program is responsible for providing this support, if needed. To provide this support, your application can have a batch job servicing remote requests to start the user-defined communications application program. This job can be created to run in any subsystem.

For more information on jobs and subsystems, see the *Work Management Guide*.

You can design your application programs so that the entire protocol resides within one job or separate jobs where each job represents a portion of the protocol.

There is a one-to-one correspondence between a job and the user-defined communications support for that job. The user-defined communications support for one job does not communicate with the user-defined communications support for another job. If two applications wish to communicate between themselves, a method such as a shared queue can be used. Also, the queue can be shared between the two (or more) jobs and the user-defined communications support for those jobs.

Figure 4-1 on page 4-2 shows how user-defined communications relate to the AS/400 system job structure and the data queue or user queue that provides the ability to communicate between your application and the user-defined communications support.

In this figure, one interactive job is running over an X.25 line (X25USA) to a system in Rochester, Minnesota, using the user-defined communications support. The link was enabled with communications handle name ROCHESTER.

The user space application programming interfaces (APIs) that the application program is using are shown, along with the programming interfaces for data and user queues and the user-defined communications support APIs.

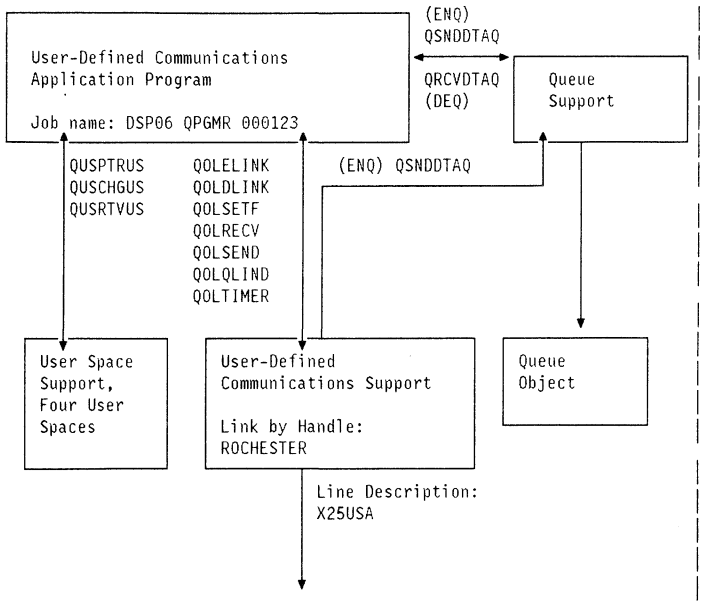


Figure 4-1. Overview of API Relationships

Figure 4-2 on page 4-2 shows two jobs, A and B. Each job is using the user-defined communications support to communicate with the networks attached to the AS/400 system by the line description. The figure shows the relationship between the different APIs and the job which is running the application program.

The lines between the jobs indicate that callable APIs that are used to communicate between the application program and the system services shown.

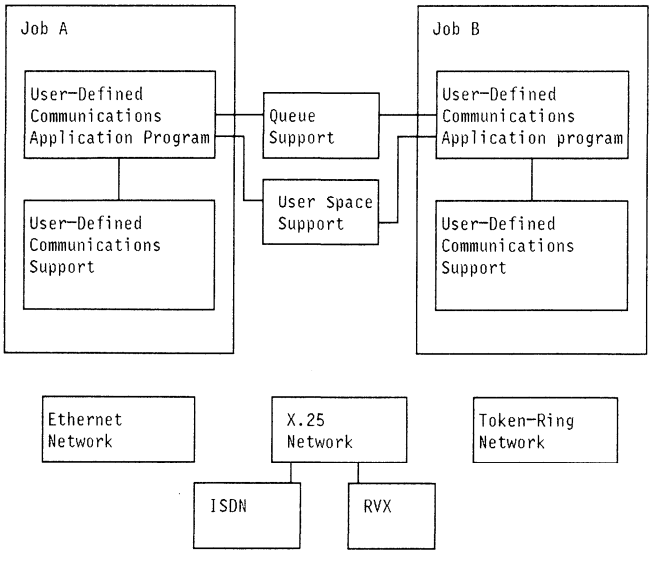


Figure 4-2. Application Programming Interface to Job Structure

The following list pertains to Figure 4-2.

- The applications use the data queue APIs, user space APIs, and user-defined communications APIs.
- An application can have more than one link enabled, and can use a separate queue for each link, or the same queue for some or all the links that it has enabled.
- The two jobs can communicate with each other using a common queue. This queue can be the same queue that is used for user-defined communications support or a different one.
- Both jobs (or any other job on the system) that has the proper authority to the user spaces, can access the user spaces.
- The user-defined communications support uses the data in the output user spaces that are created when the link is created. The application making the call to the Send Data (QOLSEND) API can fill the output buffer and descriptor, or another application can do this.
- The user-defined communications support sends data to the application through the input buffer and input descriptor that is created when the link on which the data is arriving was created. Either the application making the call to the Receive Data (QOLRECV) API retrieves the data from the input buffer and descriptor, or another application with access to the user spaces does this.
- The application supplies any communications handle (link name) to the link as long as this name is unique among all the other links that the job has enabled.
- An application can enable as many links as there are line descriptions that are supported (X.25, token-ring and Ethernet) and that are able to be varied on.
- An application is able to run over X.25 and LAN links concurrently.

Application Program Feedback

The user-defined communications support uses return and reason codes to indicate the success or failure of an operation, and provide suggested recovery information. In severe error conditions an escape message is signaled to the application program. If a severe error occurs, user-defined communications is no longer available to the application.

When the QOLSEND and QOLRECV APIs return to the application and you are running to an X.25 network, the diagnostic field is filled in. The reason code indicates whether or not the application program should look at the data returned in the diagnostic field. The diagnostic field contains additional information on the error or condition that is reported.

Synchronous and Asynchronous Operations

Most operations that an application program requests on the call to the QOLSEND API are synchronous operations. Synchronous operations involve one step, which is to call the QOLSEND API, passing the appropriate information. Synchronous operations complete when the QOLSEND API returns to the application program. The success or failure

of the operation is reported in the return and reason codes by the QOLSEND API.

Asynchronous operations do not complete when the QOLSEND API returns to the application. There are two steps for every asynchronous operation:

1. Call the QOLSEND API to initiate or request the operation.
2. Call the QOLRECV API to receive the results of the completed operation.

When the QOLSEND API returns to the application program, the request for the operation is successfully submitted. After the requested operation is complete, the user-defined communications support sends an incoming data entry (if necessary) to the queue to instruct the application program to call the QOLRECV API to receive the data. When this call to the QOLRECV API returns, the return and reason codes in the parameter list contain the success or failure of the operation. If the operation was unsuccessful due to an application template error in the user space used for output, the request data given to QOLSEND using the output buffer and descriptor is copied into the input buffer and descriptor. The offset to the template error detected is returned in the parameter list of the QOLRECV API. Asynchronous operations are only used for open connection requests, close connection requests, and resets.

For either type of operation, the application program is allowed to use the output user spaces again as soon as the call to the QOLSEND API returns.

Programming Languages

Any program written in an AS/400 system-supported language can call user-defined communications support. One consideration for choosing one language over another, is that the programming language must have the ability to set a byte field to any hexadecimal value. This does not restrict programming in the different languages, but it does make some languages more appealing than others.

Starting and Ending Communications

Relatively little configuration is required by user-defined communications support to begin communications to the network. For information on configuration, see Chapter 5, "Configuration and Queue Entries" on page 5-1.

To start communications with a network, your user-defined communications application program enables the link to the network by calling the Enable Link (QOLELINK) API. Once the link is enabled, the application program can call any of the user-defined communications support APIs, and request any of the operations supported for the link. When the application program completes communications with the network, it disables the link by calling the Disable Link (QOLDLINK) API.

Note: Enabling the link does not result in any communications activity on the network. Disabling a link may cause communications activity for X.25 links if connections are active when the link is disabled.

Using Connection Identifiers

Connection identifiers are used for connection-oriented support over X.25 networks. The connectionless connection identifiers (UCEP=1, PCEP=1) are used for local area networks. The following examples (Figure 4-3 on page 4-4 through Figure 4-14 on page 4-18) illustrate how to use connection identifiers (UCEP and PCEP). They show how the two step operations, open connection request, and close connection request relate to the UCEP and PCEP identifiers. Note the outstanding two-step operations. This is important so that the application can correctly interpret the PCEP and reuse UCEPs.

The connections in each figure refer to SVC connections, and the examples use the Receive Data Queue (QRCVDTAQ) API. The same principles apply when using PVC connections and user queues.

Example 1

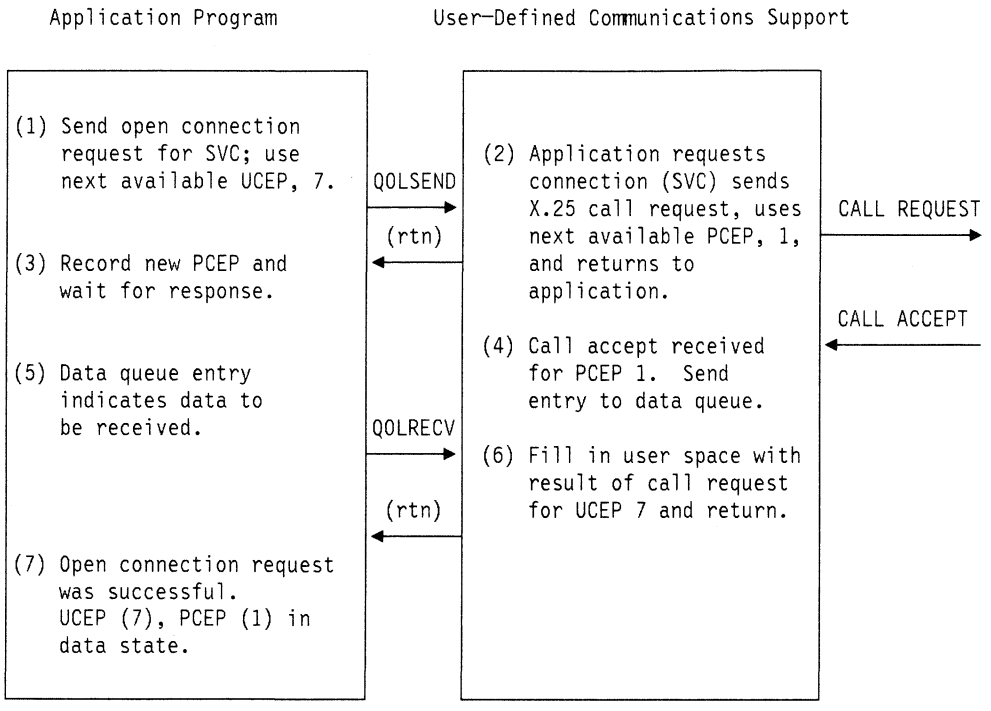


Figure 4-3. Example 1: Normal Connection Establishment

1. The application wants to open a connection, so it calls the QOLSEND API passing it the UCEP it wants to use for the connection. The application keeps track of the UCEP, PCEP pair. At this point, the UCEP=7, and the PCEP is undefined.
2. The user-defined communications support receives the request, stores the UCEP for the connection, and uses the next available PCEP, which is 1, and returns to the application, acknowledging the receipt of the request.

   The user-defined communications support validates the request and issues the X.25 call request.
3. The application records that the PCEP for UCEP=7 is 1. The UCEP=7, PCEP=1 connection is not yet active. Next, the application calls the Receive Entry From Data Queue (QRCVDTAQ) API, to wait for the incoming data entry. The application is expecting the open connection response.
4. The X.25 call accept is received for PCEP=1. To inform the application of the incoming data, an incoming data entry is sent to the data queue.
5. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to the QOLRECV API.
6. The user-defined communications support fills in the input buffer and descriptor with data for the open connection response operation, and determines the UCEP associated with the data by examining the PCEP associated with the X.25 call accept. Because the call accept was received for PCEP=1, the UCEP=7.
7. The application's call to the QOLRECV API returns with successful return and reason codes for the open connection response operation. This operation was reported for UCEP 7; the UCEP=7, PCEP=1 connection is now active.

Example 2

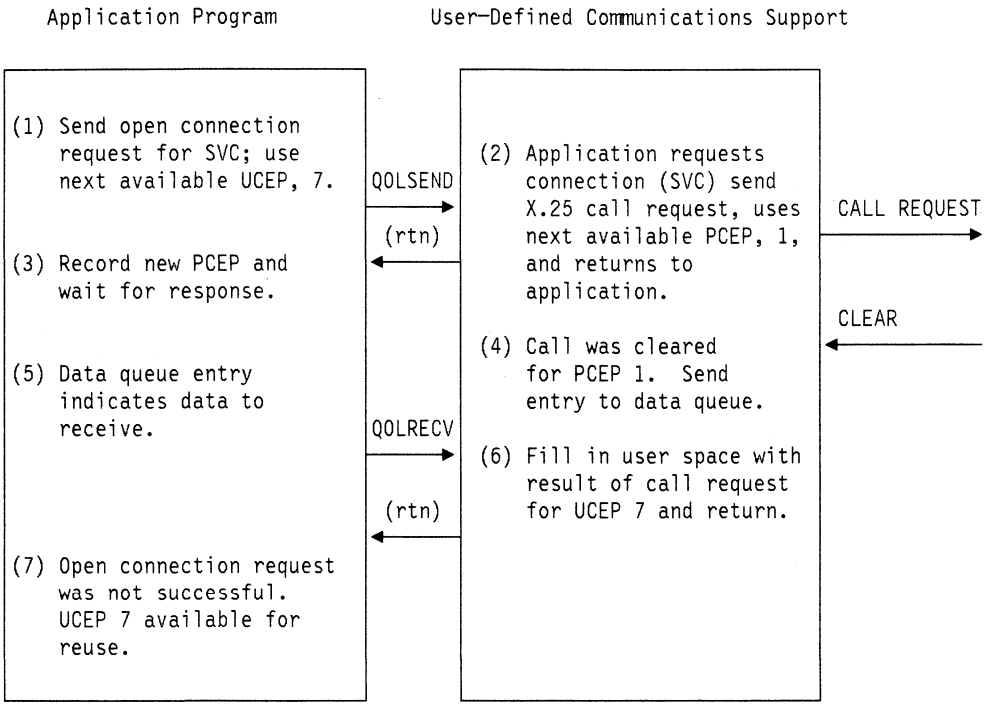


Figure 4-4. Example 2: Connection Request Cleared by Network/Remote System

1. The application wishes to open a connection, so it calls the QOLSEND API, passing it the UCEP it wants to use for the new connection. The application keeps track of the UCEP, PCEP pair. At this point, the UCEP=7, and the PCEP is undefined.
2. The user-defined communications support receives the request, stores the UCEP for the connection, and uses the next available PCEP, which is 1, and returns to the application, acknowledging the receipt of the request.

   The user-defined communications support validates the request and issues the X.25 call request.
3. The application records that the PCEP for UCEP=7 is 1, and the UCEP=7, PCEP=1 connection is not yet active. Next, the application calls the QRCVDTAQ API to wait for the incoming data entry. The application is expecting the open-connection response.
4. A clear is received for PCEP=1. To inform the application of the incoming data, an incoming data entry is sent to the data queue.
5. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to the QOLRECV API.
6. The user-defined communications support fills in the input buffer and descriptor with data for the open connection response operation, and determines the UCEP for the data by using the PCEP for which the X.25 call accept was received. Because the call was cleared for PCEP=1, the UCEP=7. The PCEP=1 is no longer active, and may be reused by the user-defined communications support.
7. The application's call to the QOLRECV API returns with unsuccessful return and reason codes for the open connection response operation. Thus for the PCEP=1, the UCEP=7. The PCEP=1 is no longer active, and the operation is for UCEP=7. Because the connection is not open, the user-defined communications support's PCEP=1 no longer implies UCEP=7, and the application's UCEP=7 may be reused.

Example 3

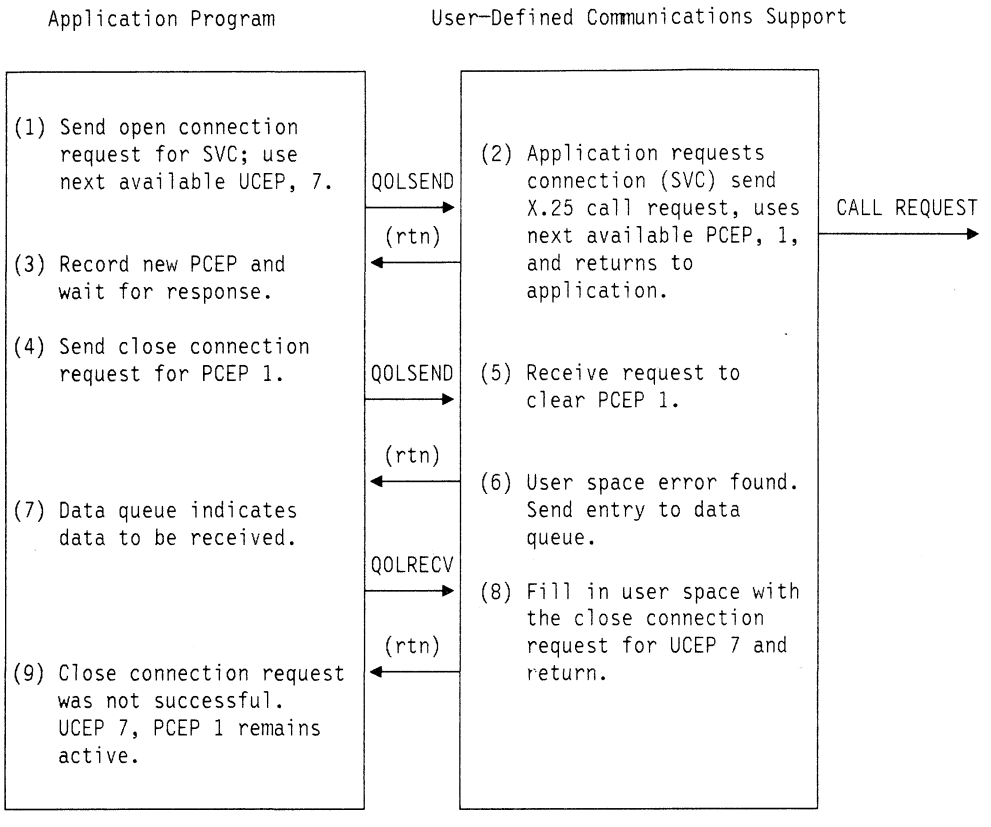


Figure 4-5. Example 3: Request to Clear Connection with Outstanding Call (Unsuccessful)

1. The application wishes to open a connection, so it calls the QOLSEND API passing it the UCEP for the new connection. The application keeps track of the UCEP, PCEP pair. At this point, the UCEP=7, and the PCEP is undefined.
2. The user-defined communications support receives the request, stores the UCEP for the connection, and uses the next available PCEP, which is 1, and returns to the application, acknowledging the receipt of the request.

   The user-defined communications support validates the request and issues the X.25 call request.
3. The application records that the PCEP for UCEP=7 is 1, and the UCEP=7, PCEP=1 connection is not yet active. Next, the application calls the QRCVDTAQ API to wait for the incoming data entry. The application is expecting the open connection response.
4. QRCVDTAQ returns to the application (the dequeue time-out value has elapsed), and the application no longer wants the UCEP=7 connection. It calls the QOLSEND API passing the PCEP=1 to identify the connection to be closed. Then the application calls the QRCVDTAQ API.
5. The user-defined communications support receives the close connection request, and returns to the application, acknowledging the receipt of the request.

   The user-defined communications support validates the request and finds an error.
6. The user space error is found. A copy of the user space, which contained an error, is passed back to the application. To inform the application of the unsuccessful close connection request, an incoming data entry is sent to the data queue.
7. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to the QOLRECV API.
8. The user-defined communications support fills in the input buffer and descriptor with data for the unsuccessful close connection request operation, and determines the UCEP associated with the data by examining the PCEP associated with the close connection. Because the close connection request was for PCEP=1, the UCEP=7.
9. The application's call to the QOLRECV API returns with unsuccessful return and reason codes for the close connection response operation. This operation is for UCEP=7. The connection UCEP=7, PCEP=1 is still in use by both the application and the user-defined communications support. The application can either correct the error and reissue the operation, or wait for the call to be accepted or rejected.

Example 4

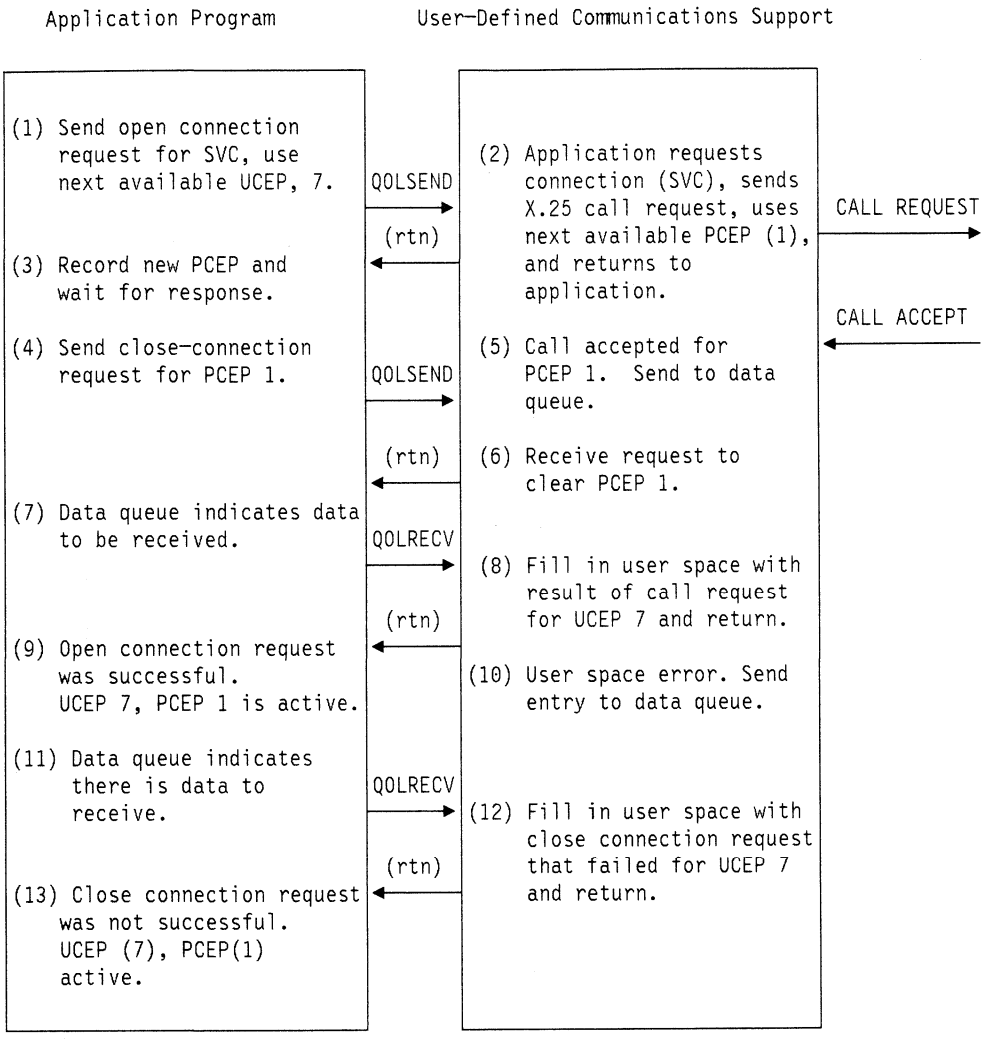


Figure 4-6. Unsuccessful Attempt to Clear Outstanding (Successful) Call

1. The application wishes to open a connection, so it calls the QOLSEND API, passing the UCEP for the new connection. The application keeps track of the UCEP, PCEP pair. At this point, the UCEP=7, and the PCEP is undefined.
2. The user-defined communications support receives the request, stores the UCEP for the connection, and uses the next available PCEP, which is 1, and returns to the application, acknowledging the receipt of the request.

   The user-defined communications support validates the request and issues the X.25 call request.
3. The application records that the PCEP for UCEP=7 is 1, and the UCEP=7, PCEP=1 connection is not yet active. Next, the application calls the QRCVDTAQ API to wait for the incoming data entry. The application is expecting the open connection response.
4. QRCVDTAQ returns to the application (the dequeue time-out value has elapsed), and the application no longer wants the UCEP=7 connection. It calls the QOLSEND API passing the PCEP=1 to identify the connection to be closed. Then the application calls the QRCVDTAQ API.
5. The X.25 call accept is received for PCEP=1. To inform the application of the incoming data, an incoming data entry is sent to the data queue.
6. The user-defined communications support receives the close connection request, and returns to the application, acknowledging the receipt of the request.
7. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to the QOLRECV API.
8. The user-defined communications support fills in the input buffer and descriptor with data for the open connection response, and determines the UCEP associated with the data by examining the PCEP for the X.25 call accept. Because the call accept was received for PCEP=1, the UCEP=7.
9. The application's call to the QOLRECV API returns with successful return and reason codes for the open connection request operation. This operation is reported

for UCEP=7; the UCEP=7, PCEP=1 connection is now active with an outstanding close connection request. The application calls the QRCVDTAQ API.

10. While processing the close connection request, the user-defined communications support detects an error in the user space. The user space that is in error is copied into the input buffer and descriptor, so the application is aware of the data in error. To inform the application of the unsuccessful close connection request, an incoming data entry is sent to the data queue.
11. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to the QOLRECV API.
12. The user-defined communications support fills the input buffer and descriptor with data for the unsuccessful close connection request operation. By using the PCEP that was requested for the close connection, the support determines the UCEP with which the data is associated. Because the close connection request was for PCEP=1, the UCEP is 7. The PCEP=1 is still active.
13. The application's call to the QOLRECV API returns with unsuccessful return and reason codes for the close connection response operation. This operation is for UCEP 7. The connection UCEP=7, PCEP=1 is still in use by both the application and the user-defined communications support. The application can either correct the error and reissue the operation, or wait for the call to be accepted or rejected.

Example 5

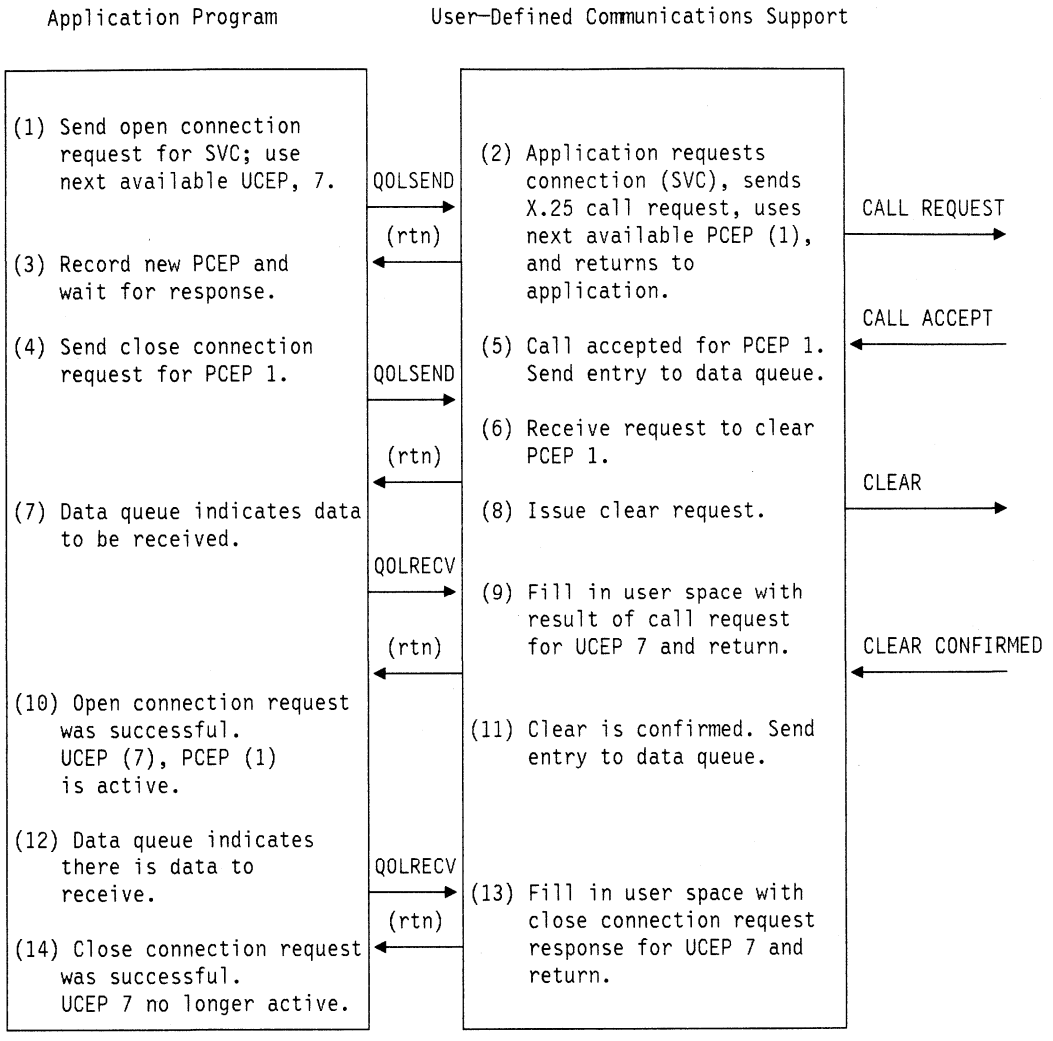


Figure 4-7. Example 5: Successful Attempt to Clear Outstanding (Successful) Call

1. The application wishes to open a connection so it calls the QOLSEND API, passing it the UCEP for the new connection. The application keeps track of the UCEP, PCEP pair. At this point, the UCEP=7, and the PCEP is undefined.
2. The user-defined communications support receives the request, stores the UCEP for the connection, and uses the next available PCEP, which is 1, and returns to the application, acknowledging the receipt of the request.

   The user-defined communications support validates the request and issues the X.25 call request.
3. The application records that the PCEP for UCEP=7 is 1, and the UCEP=7, PCEP=1 connection is not yet active. Next, the application calls the QRCVDTAQ API to wait for the incoming data entry. The application is expecting the open connection response.
4. QRCVDTAQ returns to the application (the dequeue time-out value has elapsed), and the application no longer wants the UCEP=7 connection. It calls the QOLSEND API passing the PCEP=1 to identify the connection to be closed. The application calls the QRCVDTAQ API.
5. The X.25 call-accept is received for PCEP=1. To inform the application of the incoming data, an incoming data entry is sent to the data queue.
6. The user-defined communications support receives the close connection request, and returns to the application, acknowledging the receipt of the request. The application calls QRCVDTAQ API.
7. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to QOLRECV.
8. The user-defined communications support validates the close connection request, and issues an X.25 clear request.
9. The user-defined communications support fills in the input buffer and descriptor with data for the open connection response, and determines the UCEP that the data is for by using the PCEP for the X.25 call accept.

Since the call accept was received for PCEP=1, the UCEP is 7.

10. The application's call to QOLRECV returns with successful return and reason codes for the open connection request operation. This operation is reported for UCEP=7; the UCEP=7, PCEP=1 connection is now active with an outstanding close connection request.
11. The clear confirmation is received for PCEP=1. To inform the application of the successful close connection request, an incoming data entry is sent to the data queue.
12. The application's call to the QRCVDTAQ API returns indicating there is data to receive.
13. The user-defined communications support fills the input buffer and descriptor with data for the successful close connection request operation, and determines the UCEP associated with the data by examining the PCEP that was requested for the close connection. Because the close connection request was for PCEP=1, the UCEP=7. The PCEP=1 is no longer active.
14. The application's call to the QOLRECV API returns with successful return and reason codes for the close connection response operation. This operation is for UCEP 7. The UCEP=7, PCEP=1 connection is no longer active.

Example 6

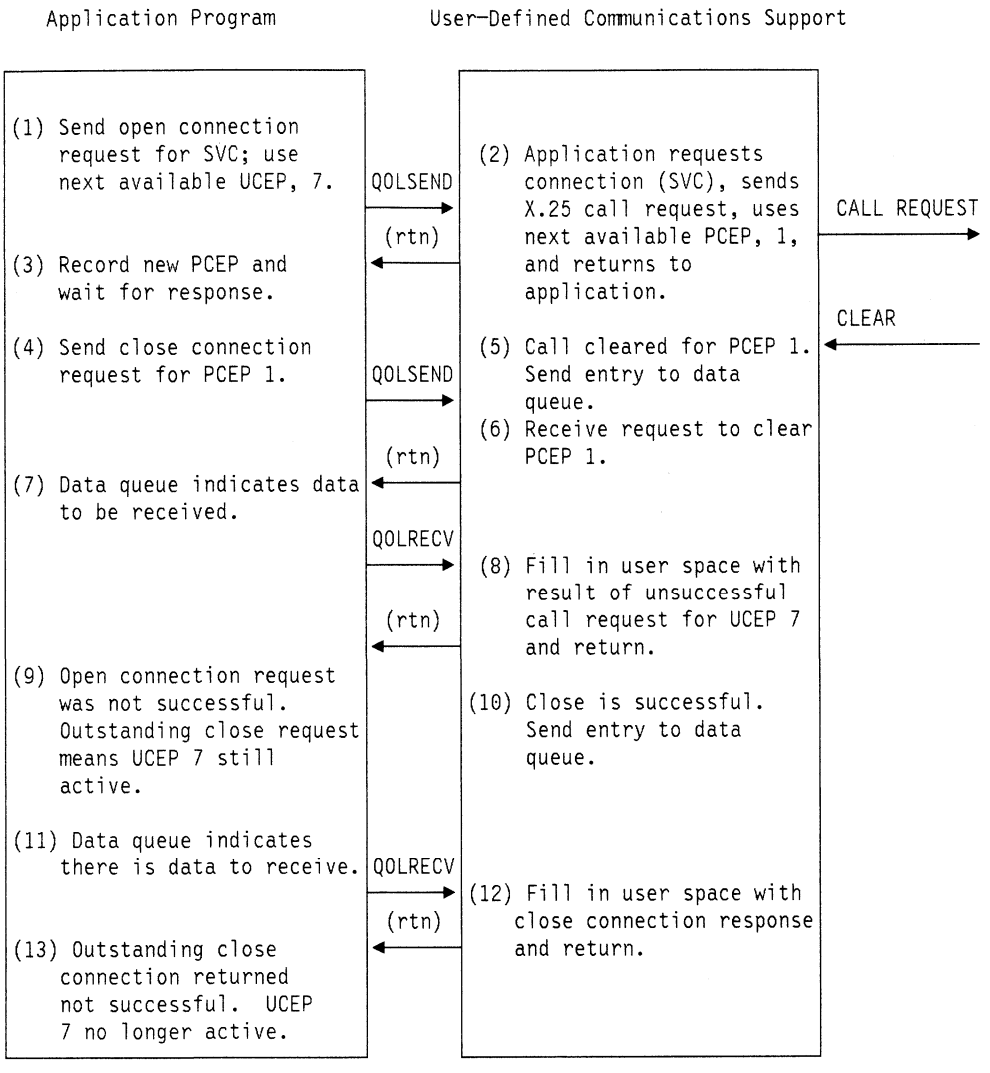


Figure 4-8. Example 6: Successful Attempt to Clear Outstanding (Unsuccessful) Call

1. The application wishes to open a connection, so it calls the QOLSEND API, passing it the UCEP for the new connection. The application keeps track of the UCEP, PCEP pair. At this point, the UCEP=7, and the PCEP is undefined.
2. The user-defined communications support receives the request, stores the UCEP for the connection, and uses the next available PCEP, which is 1, and returns to the application, acknowledging the receipt of the request.

   The user-defined communications support validates the request and issues the X.25 call request.
3. The application records that the PCEP for UCEP=7 is 1, and the UCEP=7, PCEP=1 connection is not yet active. Next, the application calls the QRCVDTAQ API to wait for the incoming data entry. The application is expecting the open connection response.
4. QRCVDTAQ API returns to the application (the dequeue time-out value has elapsed), and the application no longer wants the UCEP=7 connection. It calls the QOLSEND API passing the PCEP=1 to identify the connection to be closed. Then the application calls the QRCVDTAQ API.
5. The X.25 clear is received for PCEP=1. To inform the application of the incoming data, an incoming data entry is sent to the data queue.
6. The user-defined communications support receives the close connection request, and returns to the application, acknowledging the receipt of the request.
7. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to the QOLRECV API.
8. The user-defined communications support fills in the input buffer and descriptor with data for the open connection response, and determines the UCEP that the data is for by using the PCEP that the X.25 clear is for. Because the clear was received for PCEP=1, the UCEP is 7.
9. The application's call to the QOLRECV API returns with unsuccessful return and reason codes for the open

connection request operation. This operation is reported for UCEP=7. Because the close connection request is outstanding, the UCEP=7, PCEP=1 connection is not fully closed. The application calls the QRCVDTAQ API.

10. The close connection request is validated, but no clear is sent because the connection was cleared previously. The close is considered successful, and an entry is sent to the data queue.
11. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to the QOLRECV API.
12. The user-defined communications support fills in the input buffer and descriptor with data for the successful close connection request operation, and determines the UCEP that the data is for by using the PCEP that the close connection was requested for. Since the close connection request was for PCEP=1, and the UCEP is 7. The PCEP=1 is no longer active.
13. The application's call to the QOLRECV API returns with unsuccessful return and reason codes for the close connection response operation. This operation is for UCEP 7. The connection UCEP=7, PCEP=1 is no longer active.

Example 7

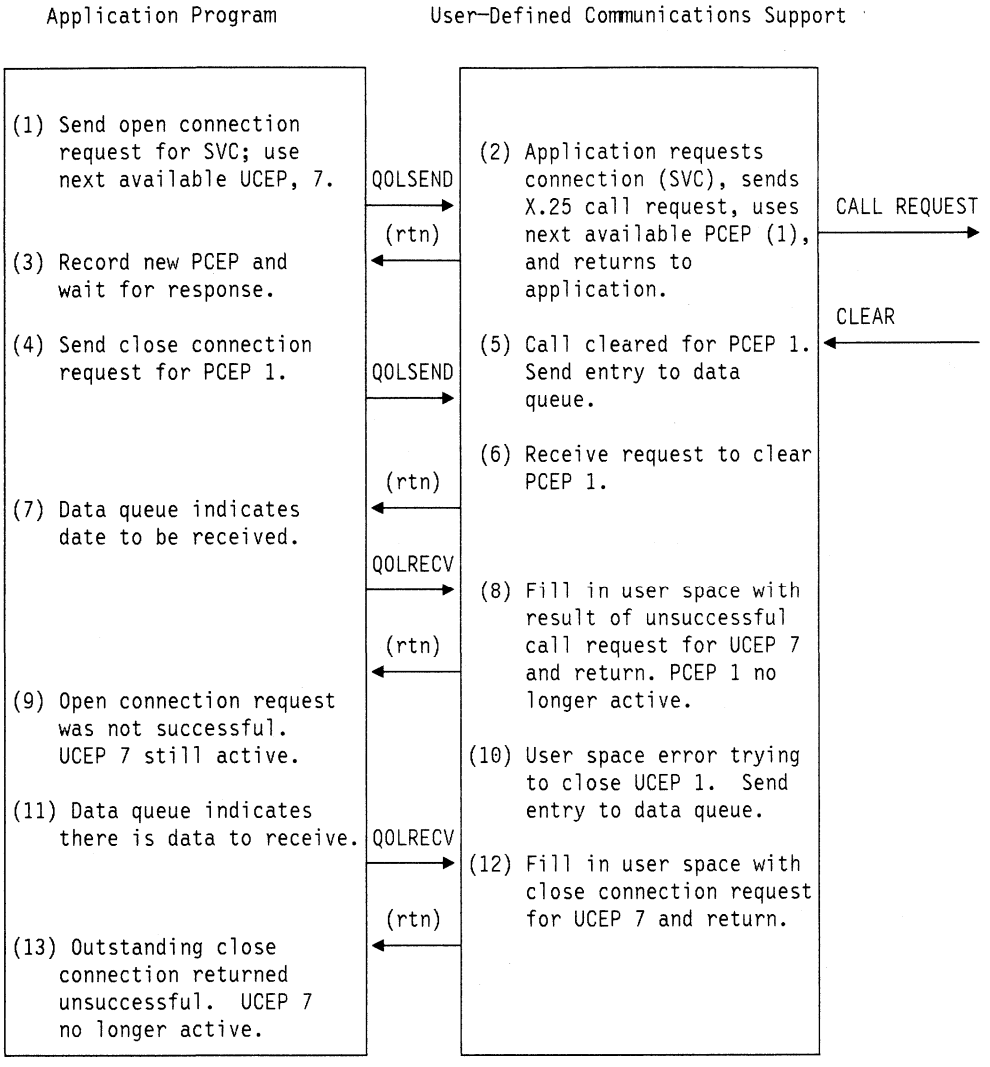


Figure 4-9. Example 7: Unsuccessful Attempt to Clear Outstanding (Unsuccessful) Call

1. The application wishes to open a connection, so it calls the QOLSEND API passing it the UCEP for the new connection. The application keeps track of the UCEP, PCEP pair. At this point, the UCEP=7, and the PCEP is undefined.
2. The user-defined communications support receives the request, stores the UCEP for the connection, and uses the next available PCEP (1); and returns to the application, acknowledging the receipt of the request.

   The user-defined communications support validates the request and issues the X.25 call request.
3. The application records that the PCEP for UCEP=7 is 1, and the UCEP=7, PCEP=1 connection is not yet active. Next, the application calls the QRCVDTAQ API to wait for the incoming data entry. The application is expecting the open connection response.
4. The application no longer wants the UCEP=7 connection. It calls the QOLSEND API passing the PCEP=1 to identify the connection to be closed. The application calls the QRCVDTAQ API.
5. The X.25 Clear is received for PCEP=1. To inform the application of the incoming data, an incoming data entry is sent to the data queue.
6. The user-defined communications support receives the close connection request, and returns to the application, acknowledging the receipt of the request.
7. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to the QOLRECV API.
8. The user-defined communications support fills in the input buffer and descriptor with data for the open connection response, and determines the UCEP that the data is for by using the PCEP that the X.25 clear is for. Because the clear was received for PCEP=1, the UCEP is 7.
9. The application's call to the QOLRECV API returns with unsuccessful return and reason codes for the open connection request operation. This operation is reported for UCEP=7. Because the close connection

request is outstanding, the UCEP=7, PCEP=1 connection is not fully closed. The application calls the QRCVDTAQ API.

10. The close connection request is validated, and an error is found in the user space. An entry is sent to the data queue.
11. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application then issues a call to the QOLRECV API.
12. The user-defined communications support fills in the input buffer and descriptor with data for the unsuccessful close connection request operation, and determines the UCEP that the data is for by using the PCEP that the close connection was requested for. Since the close connection request was for PCEP=1, the UCEP is 7. Because the connection was cleared prior to the close connection request, the PCEP=1, UCEP=7 connection is considered no longer active to the user-defined communications support.
13. The application's call to the QOLRECV API returns with unsuccessful return and reason codes for the close connection response operation. This operation is for UCEP 7. The connection UCEP=7, PCEP=1 is no longer active.

Example 1

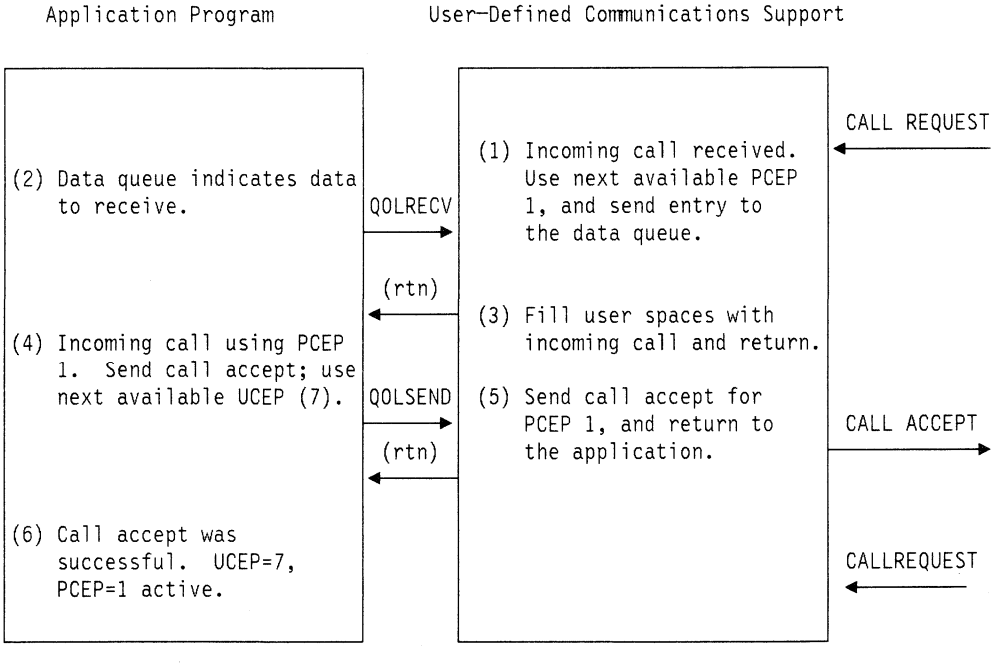


Figure 4-10. Example 1: Normal Connection Establishment

1. An incoming call request is received by the communications support, which determines if there is an application that has a filter satisfying this call request. The communications support uses the next available PCEP, which is 1, for this new connection. An entry is sent to the data queue.
2. The application has been waiting for its call to the QRCVDTAQ API to complete. The call completes indicating there is data to be received. The application calls the QOLRECV API.
3. The input buffer and descriptor are filled with the incoming call request for PCEP=1, and the QOLRECV API returns.
4. The application looks at the operation, which indicates an incoming call indication. The PCEP reported by the communications support is 1. The application chooses

Incoming Connections: The following figures show how the application program handles UCEPs and PCEPs for incoming connections.

to accept this call, and passes the UCEP to be used for this new connection. The call is made to the QOLSEND API with PCEP=1, UCEP=7.

5. The call accept is received and sent for PCEP=1. The QOLSEND API returns to the application.
6. The call accept request was successful for UCEP=7, PCEP=1. This connection is now active.

Example 2

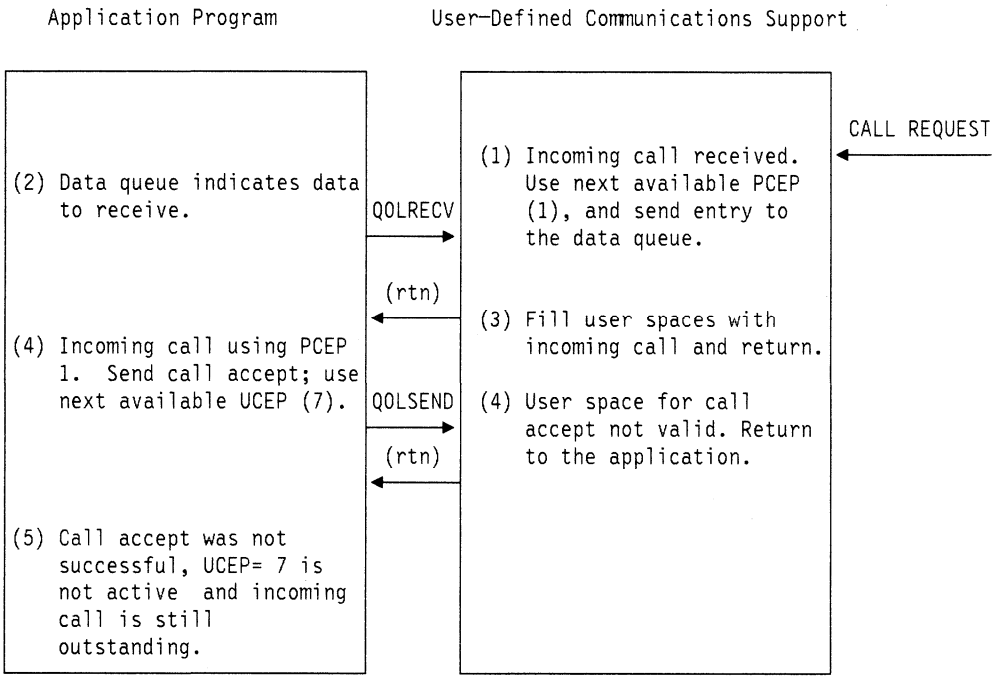


Figure 4-11. Example 2: Send Call Accept Not Valid

1. An incoming call request is received by the communications support, which determines if there is an application that has a filter satisfying this call request. The communications support uses the next available PCEP=1 for this new connection. An entry is sent to the data queue.
2. The application has been waiting for its call to the QRCVDTAQ API to complete. It does, indicating there is data to be received. The application calls the QOLRECV API.
3. The input buffer and descriptor are filled with the incoming call request for PCEP=1, and the QOLRECV API returns.
4. The application looks at the operation which indicates an incoming call indication. The PCEP reported by the communications support is 1. The application chooses to accept this call, and passes the UCEP to be used for this new connection. The call is made to the QOLSEND API with PCEP=1, UCEP=7.
5. The call accept is received and an error is found in the user space. The QOLSEND API returns to the application, reporting the error and offset. The incoming call is still outstanding for PCEP=1.

   The application checks the return and reason codes and finds that an error has occurred. The call accept was not sent and the incoming call is still waiting for a response.

Example 3

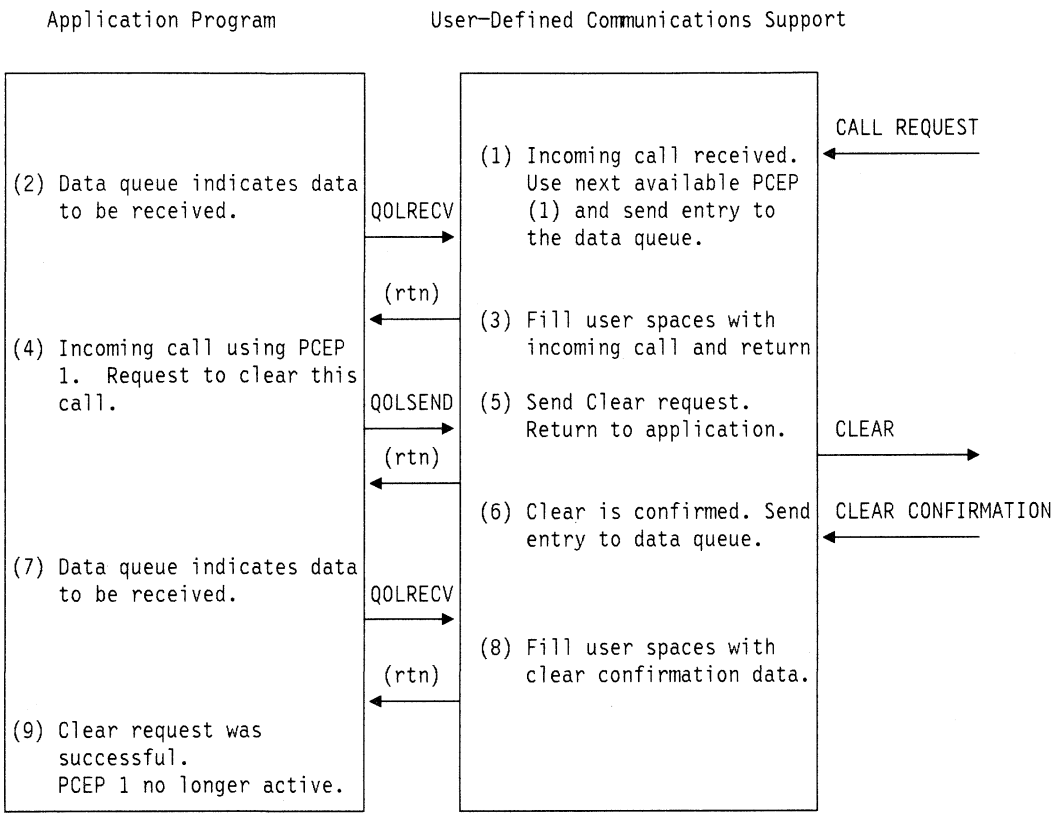


Figure 4-12. Example 3: Send Clear for Incoming Call

1. An incoming call request is received by the communications support, which determines if there is an application that has a filter satisfying this call request. The communications support uses the next available PCEP, which is 1, for this new connection. An entry is sent to the data queue.
2. The application has been waiting for its call to the QRCVDTAQ API to complete. It does, indicating there is data to be received. The application calls the QOLRECV API.
3. The input buffer and descriptor are filled for the incoming call request for PCEP = 1, and the QOLRECV API returns.
4. The application looks at the operation which indicates an incoming call indication. The PCEP reported by the communications support is 1. The application does not wish to accept the call, so the user space is filled in for a close connection request and the application calls the QOLSEND API. The application calls the QRCVDTAQ API.
5. The close connection request is received and the QOLSEND API returns to the application, acknowledging the request.

   The close connection request is validated and a clear is sent.
6. The clear confirmation is received for PCEP = 1 which has no UCEP. An incoming data entry is sent to the data queue. The application calls the QRCVDTAQ API.
7. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application calls the QOLRECV API to receive the data.
8. The input buffer and descriptor are filled in with the clear confirmation data. Since the connection was never established (and the application never assigned a UCEP to this connection), the QOLRECV API returns to the application passing a UCEP = 0.
9. The close connection request was successful. PCEP = 1 is no longer active.

Example 4

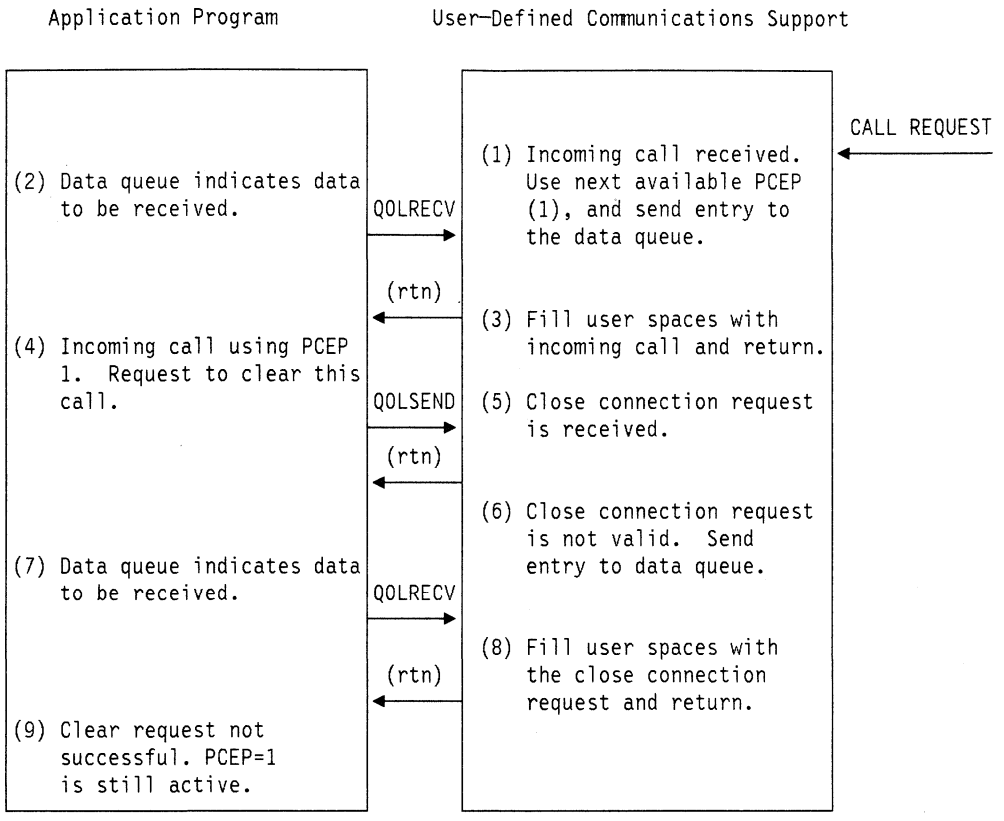


Figure 4-13. Example 4: Send Clear for Incoming Call

1. An incoming call request is received by the communications support, which determines there is an application that has a filter satisfying this call request. The communications support uses the next available PCEP=1 for this new connection. An entry is sent to the data queue.
2. The application has been waiting for its call to the QRCVDTAQ API to complete. It completes indicating there is data to be received. The application calls the QOLRECV API.
3. The input buffer and descriptor are filled for the incoming call request for PCEP=1, and the QOLRECV API returns.
4. The application looks at the operation which indicates an incoming call indication. The PCEP reported by the communications support is 1. The application does not wish to accept the call, so the user space is filled in for a close connection request and the QOLSEND API. The application calls the QRCVDTAQ API.
5. The close connection request is received and the QOLSEND API returns to the application, acknowledging the request.
6. The close connection request is validated and an error is found. An entry is sent to the data queue.
7. The application's call to the QRCVDTAQ API return, with the incoming data entry. The application calls the QOLRECV API to receive the data.
8. The input buffer and descriptor are filled in with the unsuccessful close request, and the QOLRECV API returns to the application.
9. The close connection request was not successful. PCEP=1 is still active.

Closing Connections: The following figures show how the application program closes a connection. The figures apply to both incoming and outgoing connections.

The next two figures illustrate that a close connection request never completely guarantees the connection will be closed.

Example 1

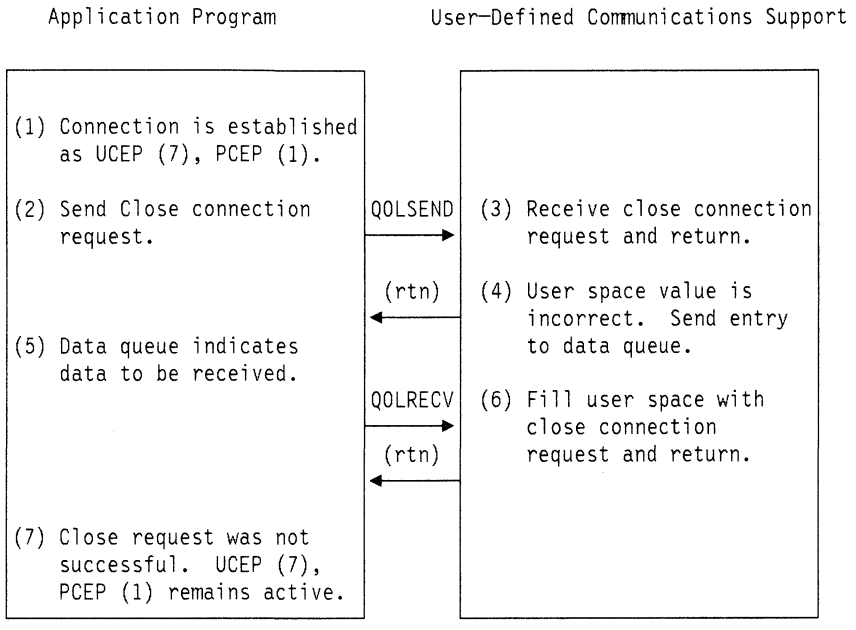


Figure 4-14. Example 1: Close Connection Request Is Not Valid

1. A connection is established with the PCEP=1, UCEP=7.
2. The application calls the QOLSEND API to close the connection. The application calls the QRCVDTAQ API.
3. The user-defined communications support receives the close connection request and returns to the application, acknowledging the receipt of the request.
4. The value in the user space is not correct. An entry is sent to the data queue.
5. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application calls the QOLRECV API to receive the data.
6. The user-defined communications support fills the input user space with data for the close connection request and determines the UCEP that the data is for by examining the PCEP that was requested for the close connection.
7. The application's call to the QOLRECV API returns with unsuccessful return and reason codes for the close connection response. This operation is for UCEP 7. The connection UCEP=7, PCEP=1 is still active.

Example 2

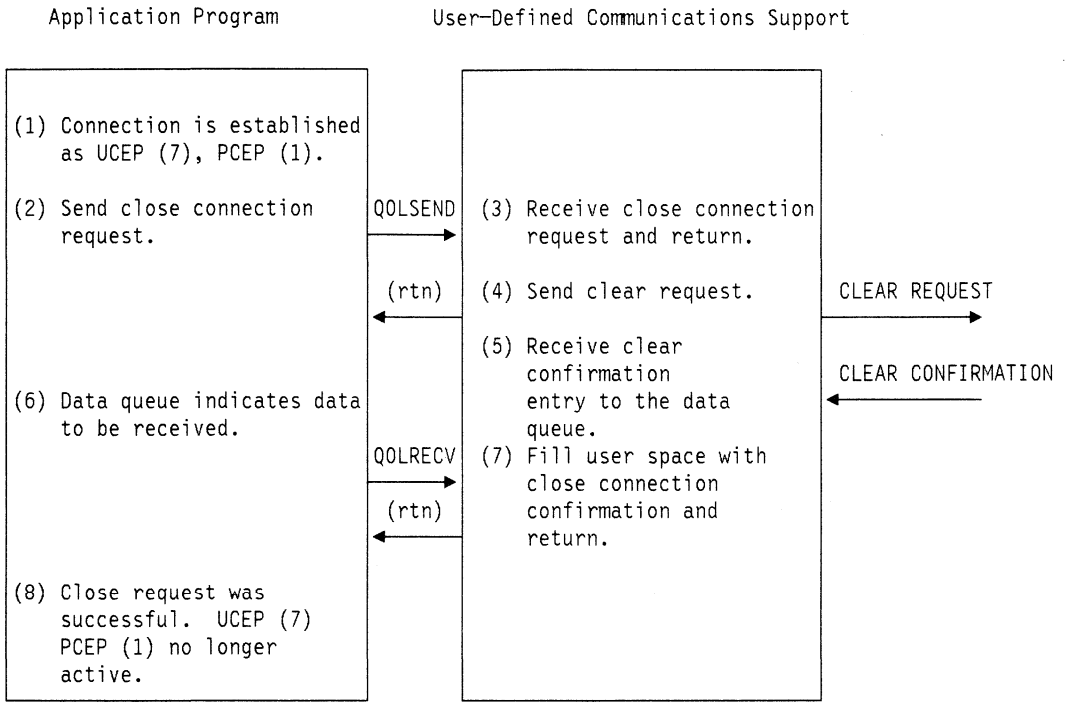


Figure 4-15. Example 2: Close Connection Request Is Valid

1. A connection is established with the PCEP=1, UCEP=7.
2. The application calls the QOLSEND API to close the connection. The application calls the QRCVDTAQ API.
3. The user-defined communications support receives the close connection request and returns to the application, acknowledging the receipt of the request.
4. The close connection request is received and the QOLSEND API returns to the application, acknowledging the request. The close connection request is validated and a clear is sent.
5. The clear confirmation is received for PCEP=1, UCEP=7. An incoming data entry is sent to the data queue.
6. The application's call to the QRCVDTAQ API returns with the incoming data entry. The application calls the QOLRECV API to receive the data.
7. The user-defined communications support fills the input user space with data for the close connection confirmation and determines the UCEP that the data is for by examining the PCEP that was requested for the close connection.
8. The application's call to the QOLRECV API returns with successful return and reason codes for the close connection response. This operation is for UCEP 7. The connection UCEP=7, PCEP=1 is no longer active.

Programming Considerations for LAN Applications

User-defined communications over LANs use connectionless (unacknowledged) service. Unacknowledged Information (UI) frames are the only frames an application program can generate.

The following tables show the frame formats of Ethernet Version 2, IEEE 802.3, and IEEE 802.5.

Figure 4-16. Ethernet Version 2 Frame Format

Frame Field	Description	Access
Preamble	Synchronization field	
Destination Address (DA)	The MAC adapter address of the remote station	X
Source Address (SA)	The MAC adapter address of the local station	
Type	The upper layer protocol used. For example, TCP/IP uses X'0800' and X'0806', SNA uses X'80D5'.	X
Information	User data	X
Frame Check Sequence (FCS)	Used for cyclic redundancy checking	

Figure 4-17. Ethernet 802.3 Frame Format

UI Frame Field	Contents & Application Access	Access
Preamble	Synchronization field	
Start Frame Delimiter (SFD)	Bit pattern that indicates the beginning of the frame	
Destination Address (DA)	The MAC adapter address of the remote station	X
Source Address (SA)	The MAC adapter address of the local station	
Length	Length of the data portion of the frame	X
Destination Service Access Point (DSAP)	Indicates the upper layer protocol the frame is for	X
Source Service Access Point (SSAP)	Indicates the upper layer protocol the frame is from	X
Control	Contains logical link control (LLC) information	
Information	User data	X
Pad	Used to pad frames less than 46 bytes	
Frame Check Sequence (FCS)	Used for cyclic redundancy checking	

Figure 4-18. Token-Ring 802.5 Frame Format

UI Frame Field	Contents & Application Access	Access
Starting Delimiter (SD)	Bit pattern that indicates the beginning of the frame	
Access Control	Contains priority, token, monitor and reserved bits **Note**: The application can only access the priority bits of this field.	X
Frame Control	Determines MAC or LLC frame	
Destination Address (DA)	The MAC adapter address of the remote station	X
Source Address (SA)	The MAC adapter address of the local station	
Routing Information	Supplied for frames intended for a system that is not locally attached to the token-ring network	X
Destination Service Access Point (DSAP)	The upper layer protocol the frame is for	X
Source Service Access Point (SSAP)	The upper layer protocol the frame is from	X
Control	Contains LLC information	
Information	User data	X
Frame Check Sequence (FCS)	Used for cyclic redundancy checking	

Figure 4-18. Token-Ring 802.5 Frame Format

UI Frame Field	Contents & Application Access	Access
Ending Delimiter	Contains error/control information	
Frame Status Field	Contains control information	

Operations: User-defined communications support defines many different operations. Not all operations are valid on all data links. The operations which are valid for LAN links are:

- X'0000' and X'0001'. These operations together represent the send- and receive-data operations for any of the LAN frames types.

Configuration

The service access point (SAP) that the application program uses to send and receive data must be configured in the line description. The 04, 06, and AA SAPs are created if *SYSGEN is specified on the CRTLINTRN or CRTLINETH commands. The 04 SAP is used by SNA, and the 06 and AA SAPs are used by TCP/IP. An application can choose to use any SAP (including SAPs defined by SNA or IEEE). The line description must be manually configured to include any other SAPs the application uses. The SAPTYPE for each SAP used must be configured as *NONSNA to be used by user-defined communications.

Although it is possible to use any SAP configurable on the AS/400 system, it is not recommended to use SNA SAPs for user-defined communications, because this may restrict the use of SNA on your AS/400 system. In the same manner, using the same SAP as other well-known protocols, such as TCP/IP, may restrict the use of these protocols or the application program on the AS/400 system.

Note: It is not possible to run an SNA application and a user-defined communications application program over the same SAP concurrently. It is possible to run a TCP/IP application and a user-defined communications application over the same SAP concurrently, provided the inbound routing information is unique among all the non-SNA applications sharing the network controller.

Inbound Routing Information

For an application program to receive data from a LAN, it must inform the communications support of how to filter the inbound data and route it to the application. This is accomplished by a program call to the Set Filter (QOLSETF) API. The fields in the incoming frame that are used to route the data are DSAP, SSAP, MAC address, and type.

The inbound routing information acts as a filter to allow the user-defined application distinguish its data from the rest of the data on the LAN. The more selective the inbound routing information is, the less chance there is that the

application will be processing unnecessary input requests. Also, more selective inbound routing information allows multiple jobs running user-defined communications applications to share the same SAP.

For example, if an application is using 92 SSAP and 92 DSAP but only talking to one remote system, it may want to set a more selective filter which would include DSAP, SSAP, and the MAC address of the remote system. Conversely, if an application accepts data on the 04 SAP from all systems sending data on any SAP, then the application would set a filter for DSAP only, indicating that it will accept all data arriving on the 04 SAP.

End-to-End Connectivity

Because user-defined communications on a LAN is connectionless, it is up to your application protocol to define a method to reach the remote systems it communicates with. There are several ways to do this. One way is to have each system configured in a database file on the AS/400 system. Each system could have a local name that the application program uses to correlate with the MAC address and routing information. LANs provide a technique to broadcast, which can be used to retrieve this information as well. An example of this is the Address Resolution Protocol (ARP) used by TCP/IP, which returns the MAC address and routing information so that a system without that information can communicate with a new remote system.

Sending and Receiving Data

Maximum Frame Size: The user-defined communications support creates a data unit size which is always large enough to contain the maximum frame size supported by any of the SAPs configured for non-SNA use, (SAPTYPE(*NONSNA)). The data unit size is returned in the parameter list on the call to the QOLELINK API. For the Ethernet (802.3) and token-ring, the maximum frame size that the application can specify is the maximum frame size allowed by the SAP that the frame is sent on. There is no minimum frame size for the Ethernet 802.3 or token-ring LANs.

Ethernet Version 2 does not define SAPs for the higher-layer protocols. Therefore, the maximum frame size is not determined by the maximum frame size for the SAP that the frame is sent on. The maximum frame size for Ethernet Version 2 is 1502 bytes. The first 2 bytes are for the type field, and the last 1500 bytes are for user data. The minimum amount of data that can be sent is 48 bytes. The first 2 bytes are for the type field, and the next 46 bytes are for user data. If the line is configured to handle both Ethernet 802.3 and Ethernet Version 2 data, the larger of the configured value or 1502 bytes is chosen and reported to the application on the data unit size parameter returned from the QOLELINK API.

If your application program attempts to send data frames that are larger or smaller than those that are supported, the output request completes with nonzero return and reason codes, and an error code is returned to the application in the diagnostic information field.

Application programs access information that is contained in the line description through the Query Line Description (QOLQLIND) API. It is best to call the QOLQLIND API after the link has been successfully enabled because the information that the QOLQLIND API passes to the application is accurate for as long as the link is enabled. The application uses the information on the frame size for the SAP to send the correct amount of data over the SAP.

Maximum Amount of Outstanding Data: Most often, the data arrives at a slightly faster rate than the application program can receive it. The communications support keeps data intended for an application so that the application can receive it. However, there is a limit to the amount of data that can be kept for the application to use later. The limit helps to avoid one system from overrunning another system's resources. When this limit is reached, all new incoming data frames for that application are discarded until the application picks up one third of the data that was stored for the application. Because the data consists of unacknowledged information frames, the higher-layer protocol within the application detects the loss of data, resends the data, or performs other recovery actions.

Each time the data limit is exceeded, the communications support creates an error log entry and puts a message in the QSYSOPR message queue, indicating that the unacknowledged service has temporarily stopped receiving incoming frames.

Ethernet to Token-Ring Conversion and Routing

The IBM 8209 Ethernet to token-ring bridge provides additional connectivity options for the AS/400 system. See the *IBM 8209 LAN Bridge Customer Information*, SA21-9994, for more details.

Performance Considerations

The application program enables connectionless traffic to enter the AS/400 system from the LAN. In the call to the QOLSETF API, the DSAP field indicates the SAP which will be activated on the AS/400 system. By activating traffic over a SAP, data is taken from the LAN and brought into the AS/400 system. Similarly, deactivating traffic over an SAP causes traffic intended for that SAP to be left at the IOP level rather than to be processed on the AS/400 system.

To minimize host processing, the SAP or SAPs that the application uses should be deactivated as soon as the application no longer wants to receive traffic for the SAP. If the link is disabled and no other applications are using the SAP(s), they are deactivated automatically by the user-defined communications support.

Protocols that use broadcast frames as a discovery technique could flood the network with messages and affect performance on all the systems attached to the network.

Programming Considerations for X.25 Applications

The user-defined communications support interface to an X.25 network is at the packet level, which is a connection-oriented level. Your application program is responsible for ensuring reliable end-to-end connectivity. **End-to-end connectivity** means that the application program can initiate, receive, and accept X.25 calls and handle network errors reported to the application, as well as send and receive data.

Your application program has access to packets that flow over switched virtual circuits (SVCs) and permanent virtual circuits (PVCs). The application can have SVC and PVC connections active concurrently. You can configure up to 64 virtual circuits on an X.25 line description, depending on the communications I/O processor used. The *X.25 Network Guide* provides more information about configuration limitations.

The Display Connection Status (DSPCNNSTS) command shows the virtual circuits that are in use by a network device, and the state of each connection. This command also displays the active inbound routing information that the application program uses to route calls.

X.25 Packet Types Supported

A **packet** is the basic unit of information transmitted through an X.25 network. The following table lists the X.25 packet types along with the type of service provided. Services for Switched Virtual Circuit (SVC) and Permanent Virtual Circuit (PVC) connections are identified as well as services that are not accessible (N/A) to an application program.

Packet Type	Application Input or Access	SVC	PVC	N/A
Data	Q,D bits of the general format identifier (GFI) **Note**: The modulus used is configured in the line description. The open connection request allows the user-defined communications support to set the actual window size used.	X	X	
Interrupt	32 bytes of data **Note**: On the AS/400 system, the X.25 packet layer provides the confirmation of the receipt of this packet. The call to the QOLSEND API does not return until the interrupt is confirmed by the remote system.	X	X	
Reset request	Cause and diagnostic codes **Note**: The application program provides the confirmation of this packet.	X	X	
Reset indication	Cause and diagnostic codes **Note**: The application program provides the confirmation of this packet.	X	X	
Reset confirmation	**Note**: User-defined communications support detects and reports reset collisions to the application on the reset confirmation.	X	X	
Incoming Call	Remote DTE, local virtual circuit, packet and window sizes, up to 109 bytes of additional facilities, up to 128 bytes of bytes of call user data	X		
Call Request	Remote DTE, local virtual circuit, packet and window sizes, up to 109 bytes of additional facilities, up to 128 bytes of bytes of call user data	X		
Call Accept	Packet and window sizes, up to 109 bytes of additional facilities	X		
Call Connected	Negotiated packet and window sizes, facilities	X		
Clear request	Cause and diagnostic codes, facilities, up to 128 bytes of clear user data	X		
Clear indication	Cause and diagnostic codes, facilities, up to 128 bytes of clear user data **Note**: The X.25 packet layer support provides the confirmation on this request.	X		
Clear confirmation	**Note**: The X.25 packet layer support provides this support.	X		
Receive Ready (RR)	**Note**: The flow of RR and RNR packets is determined by the automatic flow control field of Format I, specified in the open connection request.			X
Receive Not Ready (RNR)	**Note**: The flow of RR and RNR packets is determined by the automatic flow control field of Format I, specified in the open connection request.			X
Reject (REJ)	**Note**: This packet is not necessarily available on all networks and is not supported by the AS/400 system.			X

Packet Type	Application Input or Access	SVC	PVC	N/A
Restart Request, Indication, and Confirmation	**Note**: These packets affect all virtual circuits on the line.			X
Diagnostic	**Note**: This packet is not necessarily available on all networks and is not supported by the AS/400 system.			X
Registration Request and Confirmation	**Note**: This packet is not necessarily available on all networks and is not supported by the AS/400 system.			X

Operations: User-defined communications support defines many different operations. The X'B000' operation either initiates an X.25 SVC call request, or a request to open a PVC. By using this operation, an application program initiates an open connection request. The X'B100' operation either initiates an X.25 SVC clear request (or confirms the connection failure), or requests closing a PVC. By using this operation, an application program initiates a close connection request. The application can use the X'BF00' operation to cause the SVC or PVC connection to be reset.

The open connection request, close connection request, and reset request (or response) operations are two-step operations. See "Synchronous and Asynchronous Operations" on page 4-2 for more information on programming for two-step operations.

The X'B400' operation initiates an X.25 SVC call accept. This operation is known as a call accept operation. The X'0000' operation initiates an X.25 Data packet for a SVC or PVC connection. This operation is called a send data operation. The call accept and send data operations are one-step operations. See "Synchronous and Asynchronous Operations" on page 4-2 for more information on programming for one step operations.

The application program does not request the other available X.25 operations. These X.25 operations are inbound packets for responses from the asynchronous operations that are reported to the application in the parameter list of the QOLRECV API. The X'B201' operation indicates an incoming X.25 SVC call and is known as the call indication operation. The X'B301' operation indicates that a temporary (reset) or permanent (clear) connection failure has occurred. It is known as the connection failure indication operation. Finally, the X'0000' operation indicates incoming data. It is known as the receive data operation.

Connections

User-defined communications support allows X.25 connections over both switched and permanent virtual circuits. Your application program can have one or many connections active at once. They can be either SVC, PVC, or both. The Display Connection Status (DSPCNNSTS) command shows the state of the connection, logical channel identifier, virtual circuit type, and other information about the call. The states of the connection include activate pending, active, deactivate pending.

When the open connection request or call accept operations are not yet complete, the connection state is activate pending. Once the open connection request or call accept operations are complete with return and reason codes of zero, the connection state is active. When the close connection request is not yet complete, or if the connection is cleared by the network, but a close connection request has not been issued by the application program, the connection state is deactivate pending.

Notes:

1. The connection enters the active state when the call accept packet is sent on the network, which is independent of the application program receiving the results of the open connection request. Likewise, a connection can become completely closed (deactivated, and no longer appears on the DSPCNNSTS screen) independent of the application program receiving the results of the close connection request. This closing occurs when the application confirms the connection failure.
2. A correctly encoded close connection request will always be successful. The only time a close connection request is not successful is when the application program has coded the close connection request incorrectly. See "Using Connection Identifiers" on page 4-3 for more information.

Connection Identifiers: To differentiate between connections, user-defined communications support and an application program both use connection identifiers from the time the connection is started to the time the connection has successfully ended.

User-defined communications support assigns an identifier for each connection. This identifier is reported back to your application program as the provider connection end point (PCEP). In the same manner, your application program assigns an identifier for each connection and reports it to the communications support as the user connection end point (UCEP). This exchange of identifiers allows both the communications support and the application program to refer to a connection in a consistent manner. The UCEP and PCEP are exchanged during the open connection request during the following operations:

- A PVC is opened.
- An outgoing call is requested.
- The call indication is received and the call accept is accepted.

User-defined communications support identifies a connection only in terms of PCEP and UCEP. For example, the user-defined communications support passes information to an application program and reports the UCEP to which the information pertains. In the same manner the application program initiates requests for a connection identified by the PCEP.

User-defined communications support uses PCEPs over again as they become free. PCEPs become free when the application program receives notification that the open connection request never completed successfully, or the close connection request completed successfully. This means that PCEPs are not used over again until the application calls the QOLRECV API, which returns either the open connection request or the close connection request. Until the PCEP is freed, the connection cannot be reused.

User-defined communications support places no restrictions on the value of the UCEP, and does not verify its uniqueness. Because user-defined communications passes all incoming data and connection failure indications to the application program using the UCEP connection identifier, the application should ensure uniqueness of each UCEP. See "Using Connection Identifiers" on page 4-3 for information on how to reuse connection identifiers.

Connection Information: In order to ensure reliable end-to-end connectivity, an application program must keep track of the control information for each connection it is responsible for. Some of this control information is shown in the following list.

- State of the connection (activating, active, deactivating, reset)
- PCEP for this connection
- SVC or PVC connection indicator
- Negotiated frame sizes; maximum data unit size
- Connection is no longer active indicator or state
- Other application specific information

The application program can use the UCEP as an index into the program's data structures, which keep track of this control information.

Switched Virtual Circuit (SVC) Connectivity

Configuration: All the users of an X.25 line description share the SVCs that are configured for that line description. These users are SNA, asynchronous X.25, OSI, TCP/IP, and user-defined communications. You should define the line description with enough SVCs to accommodate all of the users of the X.25 line.

Any SVCs defined in the X.25 line description that are not in use by any controllers (including the network controller) are available to an application program. The available SVCs are distributed as they are requested by the users of the X.25 line description.

See the *X.25 Network Guide* for more information on configuring X.25 line descriptions.

For user-defined communications, the application uses an SVC when it either initiates a call, or receives an incoming call. The SVC is no longer in use when the application successfully initiates a clear request to the SVC. Like PVCs, SVCs allow only one application program to have an active connection using the virtual circuit at a time.

Inbound Routing Information: Before an application program can receive and accept an incoming call, it must first describe to the user-defined communications support the X.25 calls that should be routed to the application. The application does this by issuing a program call to the QOLSETF API, specifying the inbound routing information in the filter.

The inbound routing information that an application program specifies is the first byte of call user data called the protocol ID, or the protocol ID combined with the calling DTE address. In addition, the application specifies whether it will accept calls with fast select and reverse charging indicated. The application program can either accept or reject any calls of which it receives indications. The advantage of using filters to allow the system to reject some calls (based on protocol ID, calling DTE address, fast select, and reverse charging indicated in the incoming call) is that the application is relieved of some of the calls it would always reject.

Once the connection is active, data flows end-to-end between systems and does not need any other technique to route it to the appropriate application.

End-to-End Connectivity: End-to-end connectivity is achieved when one system initiates a call and another accepts the call. When this happens, a connection is established, and the state of the connection is active. It remains active until either one of the application programs initiates a clear request, or the network (or system) clears the connection due to an error condition.

Permanent Virtual Circuit (PVC) Connectivity

Configuration: SNA and Asynchronous X.25 controllers use PVCs on the X.25 line by configuring the controller description to logically attach to the PVC. This is not true for users of the network controller description. When a PVC is in use by an application program, the system will logically attach the network controller to the PVC. This means that any PVC defined in the X.25 line description and not attached to any controller (including the network controller) is available for use by any application that has a link enabled for the network to which the line is attached.

Because the attaching of PVCs to applications is programmable, one job can have an open connection over the PVC, end the connection, and then another job can open a different connection over the same PVC. Like SVCs, PVCs allow only one application program at a time to have an active connection using the virtual circuit.

Inbound Routing Information: By definition, the PVC does not require a call to set up a path from one system to another system. As its name suggests, this path always exists (permanent). Because there is no incoming call to route, the application has no need to set a filter for the inbound routing information. Once the application has opened the PVC, there is no other information needed for the system to route packets on the PVC to the application.

End-to-End Connectivity: The application is responsible for opening and closing PVC connections. To open a PVC, the application uses the open connection request operation, just as it does to initiate an X.25 SVC call. To close the PVC, the application uses the close connection request just as it does to clear the SVC call.

Both systems that want to communicate end-to-end must first open the virtual circuit on the local system. When the PVC is opened on the AS/400 system it is considered active and in use by the application. This is true even if the corresponding remote system doesn't have the virtual circuit active. On the AS/400 system, an open connection request always completes with return and reason codes of zero as long as the PVC is defined in the line description and is not in use by another application. There is no way to detect whether true end-to-end connectivity exists on the PVC.

If the virtual circuit is not active on both systems, and one system attempts to communicate with the other, the virtual circuit on the system with the open PVC connection is reset. An application that supports X.25 resets, sees the reset arrive as a result of the attempt to send data. In order to continue, the application responds to the reset. An application that does not support X.25 resets sees a connection failure. The application closes the PVC and opens the PVC again in order to continue to use the PVC.

Similarly, when a PVC connection is closed from one system, the other system sees a reset (if reset is supported by that application) or a connection failure if reset is not supported. If the application sees a reset, it must respond to the reset before communications can continue on that connection.

Sending and Receiving Data Packets

Data Sizes: Data units larger and smaller than the negotiated transmit packet size can be sent by an application program. Each data unit will be segmented into the appropriate packet sizes by the AS/400 system. Contiguous data larger than the negotiated packet size can also be sent. The data is divided into individual packets and sent out with the more-data indicator on. The application program should request that the data unit size be a multiple of the transmit and receive packet sizes configured in the line description. The application program sets other important values that pertain to each connection. See "X.25 SVC and PVC Output Operations" on page 6-31 for information about these values.

The values your application supplies should be carefully determined and tailored to the needs of the application. Similarly, your application uses the values returned from the system to ensure that the application does not exceed negotiated limitations.

The application uses three values to determine how to fill the user-space output buffer. These values are:

- Data unit size
- Maximum data unit assembly size
- Negotiated transmit packet size

The data unit size is the value that an application specifies when the link is enabled. The maximum data unit assembly size is the total length of contiguous input data that is assembled by the AS/400 system before passing it to the application. Contiguous data units have the more-data indicator set on in each descriptor for all the data units in the sequence except the last data unit, which has the more-data indicator set off. The application specifies the maximum data unit assembly size on the open connection request. The maximum data unit assembly size should always be greater than the data unit size to make full use of the user spaces. The negotiated transmit packet size is returned when the open connection request completes. The application uses these values together to determine how to fill in the user space output buffer.

Note: If the maximum data unit assembly size is exceeded, the data is passed up to the application with the more-data indicator on. If the connection is abnormally reset or cleared, the application may not receive the rest of the contiguous data, which was in progress during the connection failure.

If the two applications remain without exceeding the maximum data unit assembly size supported on the remote system, the system guarantees that the application receives the complete, contiguous data packet sequence.

See "Maximum Amount of Outstanding Data" on page 4-26 for related information on incoming data limitations.

Interrupts: The interrupt is a special data packet. The X.25 network imposes the restriction that a DTE cannot have more than one outstanding interrupt on any virtual call in each direction. An application program issues an interrupt by calling the QOLSEND API. The QOLSEND API does not return to the application program until the interrupt confirmation has been received. It is important to understand the interrupt confirmation procedures of the remote DTE and its implications to the local system and application.

Flow Control: The AS/400 system sends the Receive Ready (RR) and Receive Not Ready (RNR) packets on behalf of the application program. The distribution of these packets is based on the automatic flow control field in the open connection request operation. The automatic flow control (RR/RNR) is sent to prevent one system from overrunning another system with data.

When the automatic flow control value is exceeded for a connection because a remote system is sending data at a rate too fast for the local system, an RNR packet is sent on behalf of the application on that local system. Once the application on the local system receives the data, an RR is sent to allow more data to be received by the local system's communications support.

The automatic flow control value should be set high enough so that RR/RNR processing does not affect performance on the virtual circuit, and low enough that the application can process the data fast enough. If an application is coded properly, the RR and RNR processing is not noticed by the application, just as for other system users of X.25.

To avoid situations where the virtual circuit is not operational because an RNR was sent, or to avoid excessive amounts of RR and RNR processing, the application program should always attempt to receive all the data from the communications support. An application that is not coded correctly can cause another application to wait indefinitely for an RR to open the virtual circuit for communications. When the applications are coded correctly, the RR and RNR packet sequences are not noticed by the applications.

Maximum Amount of Outstanding Data: The communications support sets aside a limited amount of data for the applications it is servicing. For X.25, it is 128K for each connection. If the 128K limitation is met, an error log entry is created and the connection is cleared (SVCs) or reset (PVCs) by the system. Before this limit is reached, the AS/400 system attempts to slow the incoming data traffic by issuing an RNR on behalf of the application. An RR is sent after the application has received one-third of the amount of outstanding data.

Reset Support: When an application program initiates a reset, it is also responsible for discarding any data that the user-defined communications support has received. The user-defined communications support only discards data if the connection is closed.

AS/400 System X.25 Call Control

The X.25 support routes X.25 calls arriving to the AS/400 system primarily based on the protocol ID field. This field is the first byte of call-user data in the X.25 call packet. For more information on the X.25 support, see *X.25 Network Guide*.

Performance Considerations

The X'0000' operation is completely synchronous. This means that control is not returned to the application until all the data passed in the data units are sent and confirmed by the DCE. Some implications of this are:

- If the application sends data on a connection that has data flow turned off (the remote system sent an RNR to the local AS/400 system), a subsequent call to the QOLSEND API with operation X'0000' will not return until the remote system sends the RR to turn flow back on for the connection.
- When transmitting Interrupt packets, control is not returned to the application until the interrupt is confirmed by the remote DTE. If the remote DTE is an AS/400 system, the interrupt is confirmed by the AS/400 system X.25 packet layer support. If the network is congested, the use of Interrupt packets may cause a decrease in performance for the application.

In these situations, it may be appropriate to have one job for each connection (each virtual circuit).

Queue Considerations

An application program uses a data queue or user queue for communications between the application and the communications support. The application should create the queue prior to the call to the QOLELINK API. For more information on creating and using a queue, see the *CL Programmer's Guide* and the *Languages: System C/400 Programming RPQ P10102 User's Guide and Reference*. The link will never be fully enabled if the queue does not exist. For example, in Figure 4-19, communications is no longer available when the user-defined communications support detects that the data queue has been deleted. The same is true for user queues.

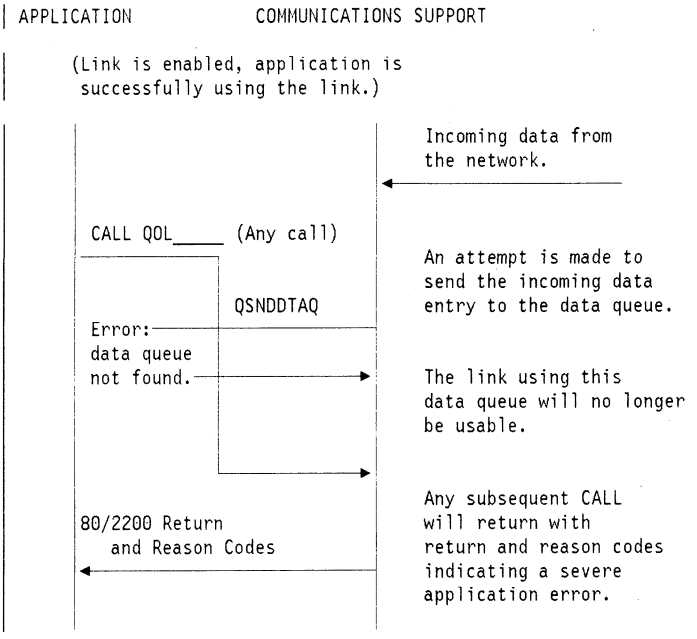


Figure 4-19. Using the Data Queue

In addition to using a queue for communications between the application and the user-defined communications support, the application can use the queue to provide communications with other applications.

If multiple processes are using the same queue, the queue can be manipulated so that each process receives queue entries based on the unique key for each application. This allows the jobs to put many kinds of entries on the queue.

For example, one key value is used for communications between the application and the system, and another key value is used for communications between the user-defined communications applications and other applications. Key values can also serve as a way to prioritize entries on the queue.

The content of the queue entries that the application defines and uses is not restricted by the user-defined communications support. User-defined communications support never attempts to receive any entries from the queue. It is the responsibility of the application to receive the entries from the queue and perform the appropriate actions for an entry based on its handle (or timer handle).

This means that it might be necessary for the application to clear the old messages from the queue if it has been used. For example, if a link is disabled, all communications entries for that link (denoted by the communications handle) prior to the disable complete entry are no longer valid.

Note: Timer support does not depend on the user-defined communications support; therefore, timer entries are still valid.

The following example shows an incoming data entry that the application receives is no longer valid because the application made a request to disable the link.

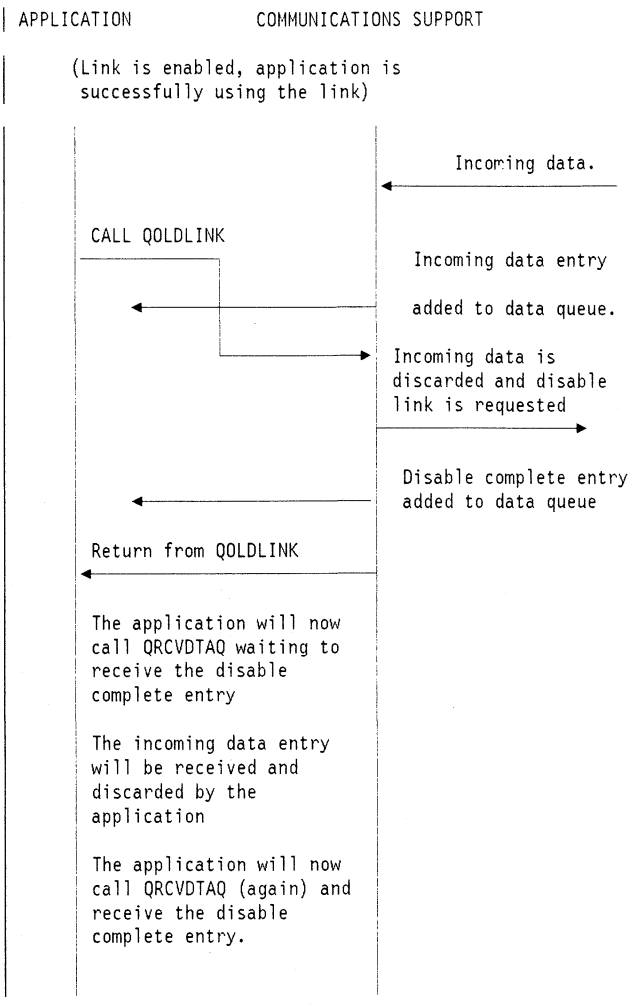


Figure 4-20. Application Disables the Link

User Space Considerations

Your application uses user space objects (*USRSPC) to hold the input and output buffers and descriptors. The AS/400 system provides APIs you can use to manipulate the user spaces.

When you use the user-defined communications support, you create the user spaces, a total of four, as part of an enable link request (the QOLELINK API). For each link, there is an input buffer, input buffer descriptor, output buffer, and output buffer descriptor. The buffers and descriptors are used to pass information to and from your application program. The buffers are used to contain user data. The descriptors are used to describe the data (length and other qualifiers). If the enable link request is not successful (return and reason codes are nonzero), the user spaces are not created.

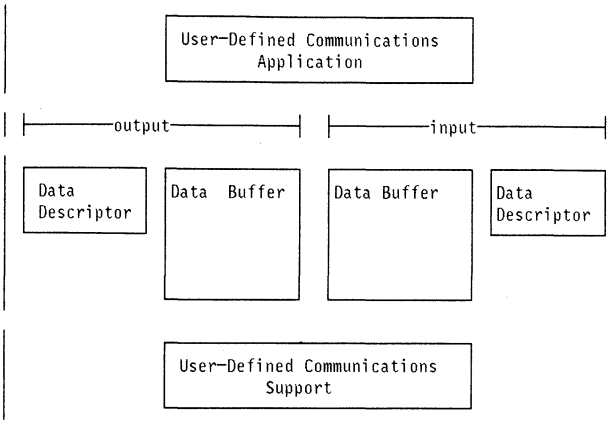


Figure 4-21. User Spaces

The buffers are divided into equally sized, contiguous sections called data units. The output buffer contains data to be sent on the network. The input buffer contains data received from the network. The size of each data unit, as well as the number of data units created, is returned from the QOLELINK API when the link is enabled.

The buffer descriptors are divided into equally sized, contiguous sections called descriptor elements. Each descriptor element describes the data in the corresponding data unit of the buffer. For example, descriptor element 1 describes the data in data unit 1 of the buffer. The size of each descriptor element is 32 bytes.

For complete and specific information about the input/output buffers, descriptors, data units, and data elements, see the sections in Chapter 6, "User-Defined Communications Support APIs" on page 6-1 describing the individual APIs.

Your application provides the library and name of the user space object that is created. The descriptive text for the object always contains the name of the job that is using these spaces. Finally, when the link is disabled (either explicitly or implicitly), these user spaces are deleted by the user-defined communications support. See "Disable Link (QOLDLINK) API" on page 6-1 for more information on disabling the link.

The application reads from the input buffer and descriptor, and writes to the output buffer and descriptor. Similarly, the user-defined communications support reads from the output buffer and descriptor and writes to the input buffer and descriptor. As soon as the call to the QOLSEND API or the QOLRECV API is complete, the application can access these user spaces.

If changes or deletions to the user spaces occur while they are in use by the user-defined communications support, a severe application error is reported to the application, and communications over the link associated with the user spaces is no longer possible.

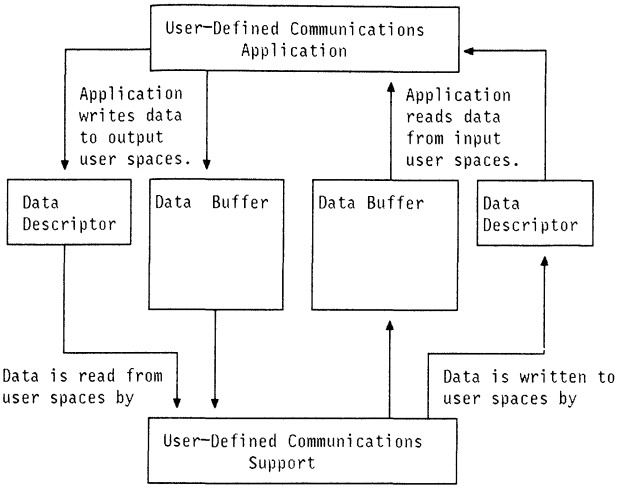


Figure 4-22. Input/Output Operations

The user-defined communications support defines logical views for the user spaces. These views are sometimes called formats. There is a format for filters, sending and receiving LAN frames, and sending and receiving X.25 packets. See "Send Data (QOLSEND) API" on page 6-27 and "Receive Data (QOLRECV) API" on page 6-14 for details on these formats.

Your application must set all the data fields required for the format. There are two types of byte fields in the buffer and descriptors, character (CHAR) and binary (BIN). Binary implies that the value is used as a numeric value. Sometimes this might be a 1-byte numeric value; for example, 12 = X'0C'. If you write the application in a language that is not capable of setting this type of binary field, the field should be declared as character and set to X'0C'. The character type contains an EBCDIC value, printable or not printable. In contrast, all parameter values are either character or 4-byte binary. See "Programming Languages" on page 4-3 for help in writing your application so that it can provide the expected input for the user-defined communications support.

The communications support never changes the output buffer; therefore, your application is responsible for initializing the buffer and descriptor for the next operation, if necessary. The data in the output buffer can also be used to help determine why a particular operation is not successful.

For performance reasons, your application should attempt to fill the output buffer as completely as possible.

Finally, for security reasons, your application chooses the library the user space object will reside in. The library can be any system library, including QTEMP. The advantage (or disadvantage) of using QTEMP for user space objects is that only the job which has enabled the links has access to the user spaces. This is because a QTEMP library exists for each job on the system. If the user space objects are in any other library, any job having authority to the library that the user spaces are in can access them.

Return Codes and Reason Codes

When control returns from a user-defined communications API to your application program, the status of the operation is located in the reason code and return code output parameters for each API.

Return codes are 4-byte numbers that determine the recovery action to take. They are grouped into the following categories:

- 00 — Operation successful, no recovery needed
- 80 — Irrecoverable error, need to disable link
- 81 — Irrecoverable error, do not need to disable link
- 82 — Recoverable error, enable link failed
- 83 — Recoverable error, see recovery actions

Reason codes are 4-byte numbers that determine what error occurred. They are grouped into the following categories:

- 0000 — No error
- 1xxx — Parameter validation or format error
- 20xx — Line, controller, or device description error
- 22xx — Data queue error
- 24xx — Buffer or buffer descriptor error
- 30xx — Link state error
- 32xx — Connection state error
- 34xx — Timer state error
- 4xxx -- Communication error
- 8xxx — Application error
- 9999 — A condition in which an Authorized Program Analysis Report (APAR) may be submitted

Note: 'x' represents any decimal number. For example, 1xxx represents the range 1000 - 1999.

For complete and specific information about the reason codes and return codes, see the sections in Chapter 6, "User-Defined Communications Support APIs" on page 6-1 describing the individual APIs.

Messages

The following messages are used to signal the success or failure of operations performed by the user-defined communications APIs:

- Information
  - CPI91F0 X.25 network error occurred.
  - CPI91F1 ISDN network error occurred.
- Escape
  - CPF91F0 Internal system error.
  - CPF91F1 User-defined communications application error.
- Diagnostic
  - CPD91F0 Error detected in program &1. Condition code is &2.
  - CPD91F1 Unexpected error detected in program &1. Condition code is &2.
  - CPD91F2 User space &1 or &3 not accessible.
  - CPD91F3 Data limit exceeded. Some data not sent.
  - CPD91F4 Error while accessing queue &1 in library &2.
  - CPD91F5 Error while accessing queue. Timer &1 canceled.
  - CPD91F6 Error occurred on line &1 while in use.
  - CPD91F7 Recovery canceled for network interface &3 or line &1.
  - CPD91F8 Error while accessing queue &1 in library &2.
  - CPD91F9 Error while enabling line &1.

Chapter 5. Configuration and Queue Entries

This chapter describes how to do the following:

- Configure user-defined communications support
- Set up the entries that user-defined communications support can send to the queue

Configuring User-Defined Communications Support

This section describes what needs to be configured before your application program can use the user-defined communications APIs. You can either use the system-supplied menus or the Control Language (CL) commands to do this configuration. For more information on using queue APIs, see the *CL Programmer's Guide* and *Languages: System C/400 Programming RPQ P10102 User's Guide and Reference*, SC09-1317.

Links

Links allow your application program to use an X.25, token-ring, or Ethernet communications line. A link is made up of the following communications objects:

- X.25, token-ring, or Ethernet line description
- Network controller description
- Network device description of type *USRDFN
- Network interface description for ISDN (X.25 only)

You need to configure the line description; user-defined communications support automatically configures a network controller and a network device description of type *USRDFN when the link is enabled. If you are using X.25 over ISDN, the network interface description must also be configured. The network interface, line, controller, and device descriptions are automatically varied on, if necessary.

Use the following commands to create or change line descriptions:

- CRTLINX25 — Create Line Description (X.25)
- CHGLINX25 — Change Line Description (X.25)
- CRTLINTRN — Create Line Description (token-ring)
- CHGLINTRN — Change Line Description (token-ring)
- CRTLINETH — Create Line Description (Ethernet)
- CHGLINETH — Change Line Description (Ethernet)

Use the following commands to create or change controller descriptions:

- CRTCTLNET — Create Controller Description (Network)
- CHGCTLNET — Change Controller Description (Network)

Use the following commands to create or change device descriptions:

- CRTDEVNET — Create Device Description (Network)
- CHGDEVNET — Change Device Description (Network)

Use the following commands to create or change network interface descriptions:

- CRTNWIISDN — Create Network Interface Description (ISDN)
- CHGNWIISDN — Change Network Interface Description (ISDN)

See the *OS/400* Communications Configuration Reference* for more information on configuring communications.

Queue

User-defined communications support uses a queue to inform your application program of some action to take or of an activity that is complete. You must create the queue before the link is enabled.

The size of each queue entry must be large enough to accommodate the user-defined communications support entries. See the following "Queue Entries" for more information on the entries that user-defined communications support can send to the queue.

Use the Create Data Queue (CRTDTAQ) command to create your data queues. Use the QUSCRTUQ and QUSDLTUQ APIs to create and delete your user queues. For more information on user queues, see the *Languages: System C/400 Programming RPQ P10203 User's Guide and Reference*, SC09-1317.

Queue Entries

This section describes the entries user-defined communications support can send to the queue.

General Format

The length of each entry is always at least 80 bytes. When using a keyed queue, however, each entry can be as large as 336 bytes, depending on the size of the key value supplied to the user-defined communications support.

Figure 5-1 shows the general format of each user-defined communications support entry.

Entry type CHAR(10)	Entry ID CHAR(2)	Entry data CHAR(68)	Key CHAR(256)
Bytes 1-10	11-12	13-80	81-336

Figure 5-1. Queue Entry General Format

Entry type: This indicates the type of entry on the queue and will be *USRDFN for all user-defined communications support entries.

Entry ID: This uniquely identifies each entry within an entry type. User-defined communications support has five entries defined:

- Enable-complete entry (entry ID = '00')
- Disable-complete entry (entry ID = '01')
- Permanent-link-failure entry (entry ID = '02')
- Incoming-data entry (entry ID = '03')
- Timer-expired entry (entry ID = '04')

Note: The entry type of *USRDFN and all associated entry IDs, either defined or undefined, are reserved for user-defined communications support. Therefore, your application program should not define entries using this entry type.

Entry data: This data is useful to your application program and varies according to the entry ID.

Key: When using a keyed queue, this is the key value supplied to the user-defined communications support.

Enable-Complete Entry

The enable-complete entry is sent to the queue when the enable link operation is complete. This entry is only sent after the Enable Link (QOLELINK) API returns to your application program with a successful return and reason code.

Note: The QOLELINK API only initiates the enabling of the link. Your application program must wait for the enable-complete entry before attempting to perform input or output on the link.

Figure 5-2 shows the format of the enable-complete entry.

*USRDFN	'00'	Communications handle	Status	Reserved	Key
Bytes 1-10	11-12	13-22	23	24-80	81-336

Figure 5-2. Enable-Complete Entry

Communications handle: The name of the link that is being enabled. Your application program supplies this name when the QOLELINK API is called.

Status: This indicates the outcome of the enable link operation. A character value of zero indicates the enable link operation was successful and I/O is now possible on this link. A character value of one indicates the enable link operation was not successful (the job log contains messages indicating the reason). The user-defined communications support disables the link when the enable link operation does not complete successfully and the disable-complete entry is not sent to the queue.

Key: The key value associated with the enable-complete entry when using a keyed queue. Your application program supplies this key value when the QOLELINK API is called. When using a non-keyed queue (indicated by supplying a key length of zero to the QOLELINK API) this field is not present.

Disable-Complete Entry

The disable-complete entry is sent to the queue when a link is successfully disabled. This entry is always the last entry sent by the user-defined communications support on this link and, therefore, provides a way for your application program to remove any enable-complete, incoming-data, or permanent-link-failure entries previously sent to the queue[1].

Figure 5-3 shows the format of the disable-complete entry.

*USRDFN	'01'	Communications handle	Reserved	Key
Bytes 1-10	11-12	13-22	23-80	81-336

Figure 5-3. Disable-Complete Entry

Communications handle: The name of the link that has been disabled. Your application program supplies this name when the QOLELINK API is called to enable the link.

Key: The key value associated with the disable-complete entry, when using a keyed queue. Your application program supplies this key value when the QOLELINK API is called to enable the link. When using a non-keyed queue (indicated by supplying a key length of zero to the QOLELINK API) this field is not present.

Permanent-Link-Failure Entry

The permanent-link-failure entry is sent to the queue when error recovery is canceled on a link. You must disable and then enable the link to recover.

Figure 5-4 shows the format of the permanent-link-failure entry.

*USRDFN	'02'	Communications handle	Reserved	Key
Bytes 1-10	11-12	13-22	23-80	81-336

Figure 5-4. Permanent-Link-Failure Entry

[1] User-defined communications support does not associate timers with links. Therefore, it is possible for a timer-expired entry to be sent to the queue after the link is disabled. Your user-defined communications application program is responsible for handling this.

Communications handle: The name of the link on which the failure has occurred. Your application program supplies this name when the QOLELINK API is called to enable the link.

Key: The key value associated with the permanent-link-failure entry, when using a keyed queue. Your application program supplies this key value when the QOLELINK API is called to enable the link. When using a non-keyed queue (indicated by supplying a key length of zero to the QOLELINK API) this field is not present.

Incoming-Data Entry

The incoming-data entry is sent to the queue when the user-defined communications support has data for your application program to receive. Your application program should call the Receive Data (QOLRECV) API to pick up the data when this entry is received.

Note: Another incoming-data entry is not sent to the queue until your application program picks up all the data from the user-defined communications support. The data available parameter on the call to the QOLRECV API indicates that the receipt of data is not complete.

Figure 5-5 shows the format of the incoming-data entry.

*USRDFN	'03'	Communications handle	Reserved	Key

Bytes 1-10 11-12 13-22 23-80 81-336

Figure 5-5. Incoming-Data Entry

Communications handle: The name of the link on which the data has come in. Your application program supplies this name when the QOLELINK API is called to enable the link.

Key: The key value associated with the incoming-data entry, when using a keyed queue. Your application program supplies this key value when the QOLELINK API is called to enable the link. When using a non-keyed queue (indicated by supplying a key length of zero to the QOLELINK API) this field is not present.

Timer-Expired Entry

The timer-expired entry is sent to the queue when a timer, previously set by your application program, ends.

Figure 5-6 shows the format of the timer-expired entry.

*USRDFN	'04'	Timer handle	User data	Key

Bytes 1-10 11-12 13-20 21-80 81-336

Figure 5-6. Timer-Expired Entry

Timer handle: The name of the expired (ended) timer. Your application program returns this name when the Set or Cancel Timer (QOLTIMER) API is called to set the timer.

User data: The data associated with the expired timer. Your application program supplies this data when the QOLTIMER API is called to set the timer.

Key: The key value associated with the timer-expired entry, when using a keyed queue. Your application program supplies this key value when the QOLTIMER API is called to set the timer. When using a non-keyed queue (indicated by supplying a key length of zero to the QOLTIMER API) this field is not present.

Chapter 6. User-Defined Communications Support APIs

User-defined communications support is made up of seven callable APIs that provide services for a user-defined communications application program. The user-defined communications APIs include the following:

Disable Link (QOLDLINK) disables one or all links.
Enable Link (QOLELINK) enables link for input and output.
Query Line Description (QOLQLIND) queries an existing line description.
Receive Data (QOLRECV) receives data from the link.
Send Data (QOLSEND) sends data from the link.
Set Filter (QOLSETF) activates or deactivates filters.
Set Timer (QOLTIMER) sets or cancels a timer.

The following sections provide detailed information about each of the user-defined communications APIs. The user-defined communications APIs are listed in alphabetic order.

Disable Link (QOLDLINK) API

Parameters

Required Parameter Group:

1	Return code	Output	Binary(4)
2	Reason code	Output	Binary(4)
3	Communications handle	Input	Char(10)
4	Vary option	Input	Char(1)

The Disable Link (QOLDLINK) API disables one or all links that are currently enabled in the job in which the application program is running. When a link is disabled, all system resources the link is using are released, the input and output buffers and descriptors for that link are deleted, and input or output on that link is no longer possible.

In addition to an application program explicitly disabling a link by calling the QOLDLINK API, user-defined communications support will implicitly disable a link in the following cases:

- When the network device associated with an enabled link is varied off from the job in which it was enabled
- When a job ends in which one or more links were enabled
- When the application program that enabled the link ends abnormally
- When the Reclaim Resource (RCLRSC) command is used
- When an unmonitored escape message is received

For each link that is successfully disabled, either explicitly or implicitly, the disable-complete entry will be sent to the data queue or user queue specified on the call to the QOLELINK API when the link was enabled. See "Disable-Complete Entry" on page 5-2 for the format of the disable-complete entry.

Required Parameter Group

Return code
OUTPUT; BINARY(4)
The recovery action to take. See "Return and Reason Codes" on page 6-2.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-2.

Communications handle
INPUT; CHAR(10)
The name of the link to disable. The special value of *ALL (left-justified and padded on the right with spaces) may be used to disable all links currently enabled in the job that the application program is running in.

Vary option
INPUT; CHAR(1)
The vary option for the network device description associated with each link being disabled. The valid values are as follows:

X'00'	Do not vary off the network device description.
X'01'	Vary off the network device description.

Return and Reason Codes

Figure 6-1. Return and Reason Codes for the QOLDLINK API

Return / Reason Code	Meaning	Recovery
0/0	Operation successful.	Continue processing.
83/1004	Vary option not valid.	Correct the vary option parameter. Then, try the request again.
83/3001	Link not enabled.	Correct the communications handle parameter. Then, try the request again.

Enable Link (QOLELINK) API

Parameters

Required Parameter Group:

1	Return code	Output	Binary(4)
2	Reason code	Output	Binary(4)
3	Data unit size	Output	Binary(4)
4	Data units created	Output	Binary(4)
5	LAN user data size	Output	Binary(4)
6	X.25 data unit size	Input	Binary(4)
7	Input buffer	Input	Char(20)
8	Input buffer descriptor	Input	Char(20)
9	Output buffer	Input	Char(20)
10	Output buffer descriptor	Input	Char(20)
11	Key length	Input	Binary(4)
12	Key value	Input	Char(256)
13	Queue name	Input	Char(20)
14	Line description	Input	Char(10)
15	Communications handle	Input	Char(10)

Optional Parameter Group:

16	Queue type	Input	Char(1)
17	Network interface description	Input	Char(10)
18	Extended operations	Input	Char(1)

The Enable Link (QOLELINK) API enables a link for input and output on a communications line. The communications line, described by the line description parameter, must be a token-ring, Ethernet, or X.25 line. The link being enabled can only be accessed within the job in which the QOLELINK API was called.

Before calling the QOLELINK API to enable a link, you must configure the following objects:

- Token-ring, Ethernet, or X.25 line description
- Data queue or user queue
- Network interface description for X.25 networks running over ISDN

See "Configuring User-Defined Communications Support" on page 5-1 for more information on configuration.

The QOLELINK API creates the input and output buffers and buffer descriptors used for the link being enabled. The network controller description and the network device description, associated with the link being enabled, are also created, if necessary. In addition, the following are varied on, if necessary.

- Line description
- Network controller description
- Network device description
- Network interface descriptions used by the line description

If the X.25 switched network interface list has multiple network interface descriptions configured, all of them can be varied on at one time. For more information on varying on network interface descriptions, refer to the *Communications Management Guide*.

When the QOLELINK API returns, your application program should examine the codes to determine the status of the link. Successful return and reason codes (both zero) indicate the link is being enabled and an enable-complete entry will be sent to the data queue or user queue specified on the call to the QOLELINK API when the enable operation completes. See "Enable-Complete Entry" on page 5-2 for more information on the enable-complete entry. Unsuccessful return and reason codes indicate the link could not be enabled and the enable-complete entry will not be sent to the data queue or user queue. "Return and Reason Codes" on page 6-4 provides more information on the QOLELINK API return and reason codes.

The queue type, network interface, and extended operations parameters make up a group parameter and if one is specified, all must be specified.

Required Parameter Group

Return code
OUTPUT; BINARY(4)
The recovery action to take. See "Return and Reason Codes" on page 6-4.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-4.

Data unit size
OUTPUT; BINARY(4)
The total number of bytes allocated for each data unit in the input and output buffers. For token-ring links, this includes user data (LAN user data size parameter), general LAN header information, and optional

routing information. For Ethernet links, this includes user data (LAN user data size parameter) and general LAN header information. For X.25 links, this includes user data (X.25 user data size parameter). For more information on the general LAN header, see Figure 6-11 on page 6-16.

Data units created
OUTPUT; BINARY(4)
The number of data units created for the input buffer and the output buffer. This parameter also specifies the number of elements created for the input buffer descriptor and the output buffer descriptor.

8 All protocols

Note: You should write your application program to avoid having to recompile should this value ever change.

LAN user data size
OUTPUT; BINARY(4)
The number of bytes allocated for token ring or Ethernet in each data unit of the input and output buffers. This does not include general LAN header information and optional routing information.

The content of this parameter is only valid when enabling a token-ring or Ethernet link.

Note: The maximum amount of token-ring or Ethernet user data that can be sent or received in each data unit is determined on a service access point basis in the line description or by the 1502 byte maximum for Ethernet Version 2 frames, and may be less than the LAN user data size. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on retrieving these values.

X.25 data unit size
INPUT; BINARY(4)
The number of bytes allocated for X.25 user data in each data unit of the input and output buffers. This is equal to the maximum amount of X.25 user data that can be sent or received in each data unit. The content of this parameter is only valid when enabling an X.25 link.

Range 512 bytes – 4096 bytes

Input buffer
INPUT; CHAR(20)
The name and library of the input buffer that the QOLELINK API creates for this link. The first 10 characters specify the name for the input buffer and the second 10 characters specify the name of an existing library that the input buffer will be created in. Both entries are left-justified. The special values of *LIBL and *CURLIB can be used for the library name.

Note: A user space object with the same name as the input buffer must not already exist in the specified library.

Input buffer descriptor
INPUT; CHAR(20)
The name and library of the input buffer descriptor that the QOLELINK API creates for this link. The first 10 characters specify the name of the input buffer descriptor and the second 10 characters specify the name of an existing library that the input buffer descriptor will be created in. Both entries are left-justified. The special values of *LIBL and *CURLIB can be used for the library name.

Note: A user space object with the same name as the input buffer descriptor must not already exist in the specified library.

Output buffer
INPUT; CHAR(20)
The name and library of the output buffer that the QOLELINK API creates for this link. The first 10 characters specify the name of the output buffer and the second 10 characters specify the name of an existing library that the output buffer will be created in. Both entries are left-justified. The special values of *LIBL and *CURLIB can be used for the library name.

Note: A user space object with the same name as the output buffer must not already exist in the specified library.

Output buffer descriptor
INPUT; CHAR(20)
The name and library of the output buffer descriptor that the QOLELINK API creates for this link. The first 10 characters specify the name of the output buffer descriptor and the second 10 characters specify the name of an existing library that the output buffer descriptor will be created in. Both entries are left-justified. The special values of *LIBL and *CURLIB can be used for the library name.

Note: A user space object with the same name as the output buffer descriptor must not already exist in the specified library.

Key length
INPUT; BINARY(4)
The key length when using a keyed data queue or user queue.

0 The data queue or user queue is not keyed.
Range 1 – 256

Key value
INPUT; CHAR(256)
The key value (left justified) when using a keyed data queue or user queue.

Queue name
INPUT; CHAR(20)
The name and library of the data queue or user queue where the enable-complete, disable-complete, permanent-link-failure, and incoming-data entries for this link will be sent. See "Queue Entries" on page 5-1 for more information about these queue entries. The first 10 characters specify the name of an

existing queue and the second 10 characters specify the library in which the queue is located. Both entries are left-justified. The special values of *LIBL and *CURLIB can be used for the library name.

Line description
INPUT; CHAR(10)
The name of the line description that describes the communications line the link being enabled will use. An existing token-ring, Ethernet, or X.25 line description must be used.

Communications handle
INPUT; CHAR(10)
The name assigned to the link being enabled. Any name complying with system object naming conventions may be used.

Optional Parameter Group

Queue type
INPUT; CHAR(1)
The type of queue you specified for the Queue name parameter.

- 'D' – Data queue
- 'U' – User queue

Network interface description
INPUT; CHAR(10)
The name of the network interface description. This value is specified if you are running X.25 and need to specify a particular network interface to use. Otherwise, this value should be set to blanks.

Note: This parameter along with the line description parameter causes only the network interface description specified to be varied on. If this value is not specified and the line description parameter contains a switched network interface list, all network interface descriptions within the list are varied on when the QOLELINK API is called.

Specifying this parameter causes only the line and the network interface that are passed to be varied on during enable processing.

Extended operations
INPUT; CHAR(1)
Indicates whether or not extended operations are supported.

Extended operations affect all connections (UCEPs, PCEPs) on the link. X'B311' and X'B111' are receive extended operations. X'B110' is a send extended operation.

- '1' – Operations supported
- '0' – Operations not supported

Return and Reason Codes

Figure 6-2 (Page 1 of 2). Return and Reason Codes for the QOLELINK API

Return / Reason Code	Meaning	Recovery
0/0	Operation successful, link enabling.	Wait to receive the enable-complete entry from the data queue or user queue before doing input/output on this link.
81/9999	Internal system error detected. Escape message CPF91F0 will be sent to the application program when this return and reason code is received.	See messages in the job log for further information. Then, report the problem using the ANZPRB command.
82/1000	User data size not valid for X.25 link.	Correct the X.25 user data size parameter. Then, try the request again.
82/1001	Key length not valid.	Correct the key length parameter. Then, try the request again.
82/1002	Queue name not valid.	Correct the queue name parameter. Then, try the request again.
82/1003	Communications handle not valid.	Correct the communications handle parameter. Then, try the request again.
82/1012	Queue type not valid.	Queue type must be D or U. Correct the queue type and try the request again.
82/1013	Extended operations value not valid.	Extended operations value must be 1 or 0. Correct the extended operations value and try the request again.
82/2000	Line name not valid or protocol is not supported.	The line name specified must be for a line of type ETHERNET, TRN, or X.25. Correct the line name and try the request again.
82/2001	Line description, network controller description, or network device description not in a valid state.	See messages in the job log indicating the affected object and recommended recovery. Do the recovery, and try the request again.

Figure 6-2 (Page 2 of 2). Return and Reason Codes for the QOLELINK API

Return / Reason Code	Meaning	Recovery
82/2002	Not authorized to the line description or network controller description.	See messages in the job log indicating the affected object and get authorization to it. Then, try the request again.
82/2003	Could not allocate the network device description.	Try the request again. If the problem continues, report the problem using the ANZPRB command.
82/2004	Could not create the network controller description or network device description.	See messages in the job log indicating the affected object and recommended recovery. Do the recovery, and try the request again.
82/2005	Could not vary on the network interface, line description, network controller description, or network device description.	See messages in the job log indicating the affected object and recommended recovery. Do the recovery, and try the request again.
82/2006	Line description not found.	Correct the line description parameter. Then, try the request again.
82/2007	Line description damaged.	Delete and re-create the line description. Then, try the request again.
82/2008	Unsupported interface. An error occurred that indicated the network interface specified cannot be associated with the line specified. For example, you specified a network interface for a token-ring or Ethernet line.	The network interface value is not correct for the line name value. Correct the configuration or your application.
82/2009	Network interface description not found.	Specify the correct network interface name and try the request again.
82/2010	Network interface description specified could not be used.	Check the network interface description for possible errors. Correct any errors and try the request again.
82/2400	An error occurred while creating the input buffer, input buffer descriptor, output buffer, or output buffer descriptor.	See messages in the job log indicating the affected object and recommended recovery. Do the recovery, and try the request again.
82/3000	Communications handle already assigned to another link that is enabled in this job.	Either disable the link that was assigned this communications handle, or correct the communications handle parameter so it does not specify a communications handle that is already assigned to a link enabled in this job. Then, try the request again.
82/3005	Line description already in use by another link that is enabled in this job.	Disable the link that is using this line description. Then, try the request again.

Error Messages

CPF91F0 E Internal system error.

Query Line Description (QOLQLIND) API

The Query Line Description (QOLQLIND) API queries an existing token-ring, Ethernet, or X.25 line description. The data received from the query is placed in the user buffer parameter.

The line description to be queried does not have to be associated with any links the application program has enabled. However, data in the line description may change after it is queried.

Parameters

Required Parameter Group:

1	Return code	Output	Binary(4)
2	Reason code	Output	Binary(4)
3	Number of bytes	Output	Binary(4)
4	User buffer	Output	Char(*)
5	Line description	Input	Char(10)
6	Format	Input	Char(1)

Optional Parameter Group:

7	Length user buffer	Input	Binary(4)
8	Bytes available	Output	Binary(4)

Required Parameter Group

Return code
OUTPUT; BINARY(4)
The recovery action to take. See "Return and Reason Codes" on page 6-13.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-13.

Number of bytes
OUTPUT; BINARY(4)
The number of bytes of data returned in the user buffer.

User buffer
OUTPUT; CHAR(*)
The buffer where the data from the query will be received. Any unused space in the buffer will be filled with X'00'. The length of this character structure is determined using Figure 6-3.

Format	Group Parameter Passed	Length of Char(*)
1	No	256
1 or 2	Yes	Specified by the length user buffer parameter.

Figure 6-3. User Buffer Format

Note: You are recommended to set the length user buffer value to a number large enough to hold the system maximum values of virtual circuits, SAPs, and group addresses with additional space left for future needs.

Line description
INPUT; CHAR(10)
The name of the line description to query. An existing token-ring, Ethernet or X.25 line description must be used.

Format
INPUT; CHAR(1)
The format of the data returned in the user buffer. The valid values are as follows:

01 Use format 01.
02 Use format 02.
See "Format of Data in the User Buffer" for more information.

Optional Parameter Group

Length user buffer
INPUT; BINARY(4)
The number of bytes available for the API to use in the user buffer parameter. The valid values are from 0 to 32,767.

Notes:

1. This parameter is required if format 2 is specified in the format parameter. It is optional if format 1 is specified.
2. If length user buffer is specified, bytes available must also be specified.
3. If additional information exists that could not be reported, the bytes available parameter will contain a larger value than the bytes returned parameter.

Bytes available
OUTPUT; BINARY(4)
The total number of bytes of available information.

Notes:

1. This parameter is required if format 2 is specified in the format parameter. It is optional if format 1 is specified.
2. If bytes available is specified, length user buffer must also be specified.
3. If the bytes available parameter contains a number larger than the bytes returned parameter, there is additional information that the application cannot access.
4. If the return code parameter is nonzero, this value is set to zero.

Format of Data in the User Buffer

The data received in the user buffer from the query is made up of two parts. The first portion starts at offset 0 from the top of the user buffer and contains general query data. The format of this data does not depend on value of the format parameter supplied to the QOLQLIND API.

Figure 6-4 (Page 1 of 2). General Query Data

Field	Type	Description
Line description	CHAR(10)	The name of the token-ring, Ethernet or X.25 line description that was queried.
Line type	CHAR(1)	The type of line description that was queried. The valid values are as follows: **X'04'** X.25 **X'05'** Token-ring **X'09'** Ethernet

Figure 6-4 (Page 2 of 2). General Query Data

Field	Type	Description
Status	CHAR(1)	The current status of the line description. The valid values are as follows: **X'00'** Varied off **X'01'** Varied off pending **X'02'** Varied on pending **X'03'** Varied on **X'04'** Active **X'05'** Connect pending **X'06'** Recovery pending **X'07'** Recovery canceled **X'08'** Failed **X'09'** Diagnostic mode **X'FF'** Unknown

The second portion of the user buffer starts immediately after the general query data and contains data specific to the type of line description that was queried. The format of this data depends on the value of the format parameter supplied to the QOLQLIND API.

Token-Ring/Ethernet Specific Data—Format 01

Figure 6-5. LAN Specific Data—Format 01

Field	Type	Description
Local adapter address	CHAR(6)	Specifies, in packed form, the local adapter address of this line. The special value of X'000000000000' indicates that the preset default address for the adapter card was configured. However, the line description must be varied on before this address can be retrieved.
Line speed	CHAR(1)	The speed of this line. The valid values are as follows: **X'01'** 4 megabits/second **X'02'** 10 megabits/second **X'03'** 16 megabits/second
Line capability	CHAR(1)	The capability of this line. The valid values are as follows: **X'00'** Token-ring **X'01'** Ethernet Version 2 **X'02'** Ethernet 802.3 **X'03'** Both Ethernet Version 2 and Ethernet 802.3
Line frame size	BINARY(2)	The maximum frame size possible on this line.
Ethernet Version 2 frame size	BINARY(2)	The maximum size for Ethernet Version 2 frames. This will be 1502 if the line is capable of Ethernet Version 2 traffic. Otherwise, it will be zero.
Number of SSAPs	BINARY(2)	The number of source service access points (SSAPs) configured for this line.
Note: The following 3 rows are repeated for each SSAP configured for this line.		
SSAP	CHAR(1)	The configured source service access point.
SSAP type	CHAR(1)	The SSAP type. The valid values are as follows: **X'00'** Non-SNA SSAP **X'01'** SNA SSAP
SSAP frame size	BINARY(2)	The maximum frame size allowed on this SSAP.
Number of group addresses.	BINARY(2)	The number of group addresses configured for this line. **Note**: This will always be zero for a token-ring line description.
Note: The following row is repeated for each group address configured for this line.		
Group address	CHAR(6)	Specifies a group address, in packed form.

Token-Ring/Ethernet Specific Data—Format 02

Figure 6-6 (Page 1 of 2). LAN Specific Data—Format 02

Field	Type	Description
Local adapter address	CHAR(6)	Specifies, in packed form, the local adapter address of this line. The special value of X'000000000000' indicates that the preset default address for the adapter card was configured. However, the line description must be varied on before this address can be retrieved.
Line speed	CHAR(1)	The speed of this line. The valid values are as follows: **X'01'** 4 megabits/second **X'02'** 10 megabits/second **X'03'** 16 megabits/second
Line capability	CHAR(1)	The capability of this line. The valid values are as follows: **X'00'** Token-ring **X'01'** Ethernet Version 2 **X'02'** Ethernet 802.3 **X'03'** Both Ethernet Version 2 and Ethernet 802.3
Line frame size	BINARY(2)	The maximum frame size possible on this line.
Ethernet Version 2 frame size	BINARY(2)	The maximum size for Ethernet Version 2 frames. This will be 1502 if the line is capable of Ethernet Version 2 traffic. Otherwise, it will be zero.

Figure 6-6 (Page 2 of 2). LAN Specific Data—Format 02

Field	Type	Description
Functional address field	CHAR(6)	The hexadecimal functional address configured for the line. An address of X'000000000000' indicates there are no functional addresses configured on this line description.
Note: For additional information on functional addresses, refer to the *Token-Ring Architecture Reference*, SC30-3374.		
Number of group addresses	BINARY(2)	The number of group addresses configured for this line. This value is valid for Ethernet line descriptions only.
Offset to group addresses	BINARY(2)	Offset within this structure to the array of group addresses
Number of SSAPs	BINARY(2)	The number of SSAPs configured for this line.
Offset to SSAPs	BINARY(2)	Offset within this structure to the array of SSAPs
Reserved	CHAR(*)	Reserved for extension
Note: The following row is duplicated by the number of group addresses.		
Group address	CHAR(6)	Specifies a group address, in packed form.
Note: The following three rows are duplicated by the number of SSAPs.		
SSAP	CHAR(1)	The configured source service access point.
SSAP type	CHAR(1)	The SSAP type. The valid values are as follows: **X'00'** Non-SNA SSAP **X'01'** SNA SSAP
SSAP frame size	BINARY(2)	The maximum frame size allowed on this SSAP.

X.25 Specific Data—Format 01

Figure 6-7 (Page 1 of 2). X.25 Specific Data—Format 01

Field	Type	Description
Local network address length	CHAR(1)	Specifies, in hexadecimal, the number of binary coded decimal (BCD) digits in the local network address.
Local network address	CHAR(9)	Specifies, in BCD, the local network address of this line.
Extended network addressing	CHAR(1)	Specifies whether network addressing is extended to permit the use of 17 digits in an address. The valid values are as follows: **X'01'** Network addresses may be up to 15 digits **X'02'** Network addresses may be up to 17 digits
Address insertion	CHAR(1)	Specifies whether the system inserts the local network address in call request and call accept packets. The valid values are as follows: **'Y'** The local network address is inserted in call request and call accept packets. **'N'** The local network address is not inserted in call request and call accept packets.
Modulus	CHAR(1)	The X.25 modulus value. The valid values are as follows: **X'01'** Modulus 8 **X'02'** Modulus 128
X.25 DCE support	CHAR(1)	Specifies whether the system communicates using the integrated X.25 DCE support. This allows the system, acting as the DCE, to communicate with another system without going through an X.25 network. The valid values are as follows: **X'01'** The system does not communicate via the X.25 DCE support **X'02'** The system does communicate via the X.25 DCE support
Transmit maximum packet size	BINARY(2)	The transmit maximum packet size configured for this line.
Receive maximum packet size	BINARY(2)	The receive maximum packet size configured for this line.

Figure 6-7 (Page 2 of 2). X.25 Specific Data—Format 01

Field	Type	Description
Transmit default packet size	BINARY(2)	The transmit default packet size configured for this line.
Receive default packet size	BINARY(2)	The receive default packet size configured for this line.
Transmit default window size	BINARY(1)	The transmit default window size configured for this line.
Receive default window size	BINARY(1)	The receive default window size configured for this line.
Number of logical channels	BINARY(2)	The number of logical channels configured for this line.
Note: The following 4 rows are repeated for each logical channel configured for this line		
Logical channel group number	CHAR(1)	The logical channel group number. This together with the logical channel number makes up the logical channel identifier.
Logical channel number	CHAR(1)	The logical channel number. This together with the logical channel group number makes up the logical channel identifier.
Logical channel type	CHAR(1)	The logical channel type. The valid values are as follows: **X'01'** Switched virtual circuit (SVC). **X'02'** Permanent virtual circuit (PVC) that is eligible for use by a network controller. **Note**: This does not necessarily mean that this PVC is available for use. Another job running on the network controller attached to this line may already have this PVC in use. **X'22'** PVC that is not eligible for use by a network controller. For example, a PVC that is already attached to an asynchronous controller description.
Logical channel direction	CHAR(1)	The direction of calls allowed on the logical channel. The valid values are as follows: **X'00'** Not applicable (PVC logical channel). **X'01'** Only incoming calls are allowed on this logical channel. **X'02'** Only outgoing calls are allowed on this logical channel. **X'03'** Both incoming and outgoing calls are allowed on this logical channel.

X.25 Specific Data—Format 02

Figure 6-8 (Page 1 of 3). X.25 Specific Data—Format 02

Field	Type	Description
Local network address length	CHAR(1)	Specifies, in hexadecimal, the number of binary coded decimal (BCD) digits in the local network address.
Local network address	CHAR(9)	Specifies, in BCD, the local network address of this line.
Extended network addressing	CHAR(1)	Specifies whether network addressing is extended to permit the use of 17 digits in an address. The valid values are as follows: **X'01'** Network addresses may be up to 15 digits **X'02'** Network addresses may be up to 17 digits
Address insertion	CHAR(1)	Specifies whether the system inserts the local network address in call request and call accept packets. The valid values are as follows: **'Y'** The local network address is inserted in call request and call accept packets. **'N'** The local network address is not inserted in call request and call accept packets.
Modulus	CHAR(1)	The X.25 modulus value. The valid values are as follows: **X'01'** Modulus 8 **X'02'** Modulus 128
X.25 DCE support	CHAR(1)	Specifies whether the system communicates using the integrated X.25 DCE support. This allows the system, acting as a DCE, to communicate with another system without going through an X.25 network. The valid values are as follows: **X'01'** The system does not communicate via the X.25 DCE support **X'02'** The system does communicate via the X.25 DCE support

Figure 6-8 (Page 2 of 3). X.25 Specific Data—Format 02

Field	Type	Description
Transmit maximum packet size	BINARY(2)	The transmit maximum packet size configured for this line.
Receive maximum packet size	BINARY(2)	The receive maximum packet size configured for this line.
Transmit default packet size	BINARY(2)	The transmit default packet size configured for this line.
Receive default packet size	BINARY(2)	The receive default packet size configured for this line.
Transmit default window size	BINARY(1)	The transmit default window size configured for this line.
Receive default window size	BINARY(1)	The receive default window size configured for this line.
Number of logical channels	BINARY(2)	The number of logical channels configured for this line.
Maximum frame size	BINARY(2)	The maximum frame size configured in the line description. The valid values are as follows: • 1024 • 2048 • 4096
ISDN interface	CHAR(1)	Indicates if the line uses an ISDN interface. The valid values are as follows: **X'00'** X.25 line does not run over an ISDN interface. **X'01'** X.25 line runs over an ISDN interface.
Note: The following section applies only if the ISDN interface is specified as X'01'. The sections of format 02 on the call direction field to the offset to logical channel array field are not meaningful if an ISDN interface is not used and will return zeros in these fields if an ISDN interface is not specified.		
Call direction	CHAR(1)	The direction of the ISDN call. The valid values are as follows: **X'00'** Incoming switched call **X'01'** Outgoing switched call **X'02'** Either a nonswitched call or not ISDN-capable.
Note: The following fields are only meaningful if the line description is switched.		
Length of call ID information	BINARY(2)	Length includes type and plan, as described below, and the call identify information element.
Type of number and numbering plan	BINARY(1)	Type and plan as represented by the following bit sequence: tttt pppp, where tttt equals the category of the calling number and pppp equals the numbering plan identification used when the calling party number was created. **Type '0000 xxxx'** Unknown number **Type '0001 xxxx'** International number **Type '0010 xxxx'** National number **Type '0011 xxxx'** Network specific number **Type '0100 xxxx'** Subscriber number **Type '0110 xxxx'** Abbreviated number **Type '0111 xxxx'** Reserved for extension **Plan 'xxxx 0000'** Unknown **Plan 'xxxx 0001'** ISDN/telephony numbering plan **Plan 'xxxx 0011'** Data numbering plan **Plan 'xxxx 0100'** Telex** numbering plan **Plan 'xxxx 1000'** National standard numbering plan **Plan 'xxxx 1001'** Private numbering plan **Plan 'xxxx 1111'** Reserved for extension **Note**: Refer to CCITT Recommendation Q.931 for more information.
Reserved	BINARY(1)	Reserved for extension.
Call ID digits	CHAR(128)	Calling party number of remote system received off the D-channel, specified in IA5 code (ASCII).
Length of sub-address information	BINARY(2)	Length includes type, odd-even indicator, and the subaddress information element. Values can range from X'0001' to X'00FF'. The user specified subaddress is restricted to 20 bytes.

Figure 6-8 (Page 3 of 3). X.25 Specific Data—Format 02

Field	Type	Description
Type of subaddress and odd-even indicator	BINARY(1)	Type and odd-even indicator as represented by the following bit sequence: tttt ixxx, where tttt equals the type of subaddress and i equals whether the address has an even or odd number of digits. **Type '0000 xxxx'** NSAP **Type '0010 xxxx'** User specified **Type remaining** Reserved **Plan 'xxxx 0xxx'** Even number of address digits **Plan 'xxxx 1xxx'** Odd number of address digits **Note**: Refer to CCITT Recommendation Q.931 for more information.
Reserved	BINARY(1)	Reserved for extension.
Subaddress	CHAR(128)	Calling party subaddress information, received from the D-channel, specified in the IA5 code set (a superset of ASCII).
Offset to logical channel array	BINARY(2)	Offset within this structure to the array of logical channels
Reserved	CHAR(*)	Reserved for extension
Note: The following 5 rows are repeated for each logical channel configured for this line. This section is not specific to ISDN interfaces.		
Logical channel group number	CHAR(1)	The logical channel group number. This together with the logical channel number makes up the logical channel identifier.
Logical channel number	CHAR(1)	The logical channel number. This together with the logical channel group number makes up the logical channel identifier.
Logical channel type	CHAR(1)	The logical channel type. The valid values are as follows: **X'01'** Switched virtual circuit (SVC). **X'02'** Permanent virtual circuit (PVC) that is eligible for use by a network controller. **Note**: This does not necessarily mean that this PVC is available for use. Another job running on the network controller attached to this line may already have this PVC in use.
Type of calls allowed	CHAR(1)	Types of calls supported on the logical channel. The valid values are as follows: **X'00'** Not applicable (PVC logical channel). **X'01'** Only incoming calls are allowed on this logical channel. **X'02'** Only outgoing calls are allowed on this logical channel. **X'03'** Both incoming and outgoing calls are allowed on this logical channel.
Availability	CHAR(1)	Specifies whether the virtual circuit is available or currently is in use. The valid values are as follows: **X'00'** Available **X'01'** In use

Return and Reason Codes

Figure 6-9. Return and Reason Codes for the QOLQLIND API

Return / Reason Code	Meaning	Recovery
00/0000	Operation successful.	Continue processing. **Notes**: 1. When calling QOLQLIND (specifying an X.25 line description, format 1, and not specifying group parameters), up to 54 logical channels can be contained in the user buffer because it is limited to a size of 256 bytes. To increase the size of the user buffer so that it is sufficient to contain all of the logical channels, the group parameters should be used. To determine if there are more than 54 logical channels configured, use the Display Line Description (DSPLIND) command. 2. The application should check to ensure that the bytes available value returned is less than or equal to the bytes returned value. If so, there is additional information that the application may want to receive. To receive this information, the application must re-issue the call, specifying the length user buffer equal to or greater than the bytes available value.
81/9999	Internal system error detected. Escape message CPF91F0 will be sent to the application program when this return and reason code is received.	See messages in the job log for further information. Report the problem using the ANZPRB command.
83/1005	Format not valid.	Correct the format parameter. Try the request again.
83/1014	Length user buffer value not valid. This value cannot be negative.	Correct the length user buffer value to a zero or a positive value less than 32K and try the operation again.
83/1020	Group parameters not valid.	All parameters within the group must be specified. Correct the parameter list and try the request again.
83/1021	Required parameter not specified.	Format 2 was requested and the required group parameters (length user buffer and bytes available) were not specified. Correct the parameter list and try the request again.
83/1998	User buffer parameter too small.	Either the length user buffer value is negative or it contains a positive value and the system was not able to put the data into the user buffer provided by the application. Correct the application and try the request again.
83/2000	Line description not configured for token-ring, Ethernet, or X.25.	Correct the line description parameter. Try the request again.
83/2002	Not authorized to line description.	Get authorization to the line description. Try the request again.
83/2006	Line description not found.	Correct the line description parameter. Try the request again.
83/2007	Line description damaged.	Delete and re-create the line description. Try the request again.

Error Messages

CPF91F0 E Internal system error.

Receive Data (QOLRECV) API

Parameters

Required Parameter Group:

1	Return code	Output	Binary(4)
2	Reason code	Output	Binary(4)
3	Existing user connection end point ID	Output	Binary(4)
4	New provider connection end point ID	Output	Binary(4)
5	Operation	Output	Char(2)
6	Number of data units	Output	Binary(4)
7	Data available	Output	Char(1)
8	Diagnostic data	Output	Char(40)
9	Communications handle	Input	Char(10)

The Receive Data (QOLRECV) API performs an input operation on a link that is currently enabled in the job in which the application program is running. The type of data received is returned in the operation parameter. The data itself, is returned in the input buffer that was created when the link was enabled. For X'0001' operations, a description of that data is also be returned in the input buffer descriptor that is created when the link was enabled.

The QOLRECV API can receive different types of data depending on the type of communications line the link is using. See "LAN Input Operations" on page 6-16 for more information on the types of data that can be received on links using a token-ring or Ethernet communications line. See "X.25 SVC and PVC Input Operations" on page 6-18 for more information on the types of data that can be received on links using an X.25 communications line.

Note: The QOLRECV API should only be called when the user-defined communications support has data available to be received. This is indicated either by an incoming-data entry on the data queue or user queue, or by the data available parameter on the QOLRECV API.

Required Parameter Group

Return code
OUTPUT; BINARY(4)
The recovery action to take. See "Return and Reason Codes" on page 6-23.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-23.

Existing user connection end point ID
OUTPUT; BINARY(4)
The user connection end point (UCEP) ID that the data was received on. For links using a token-ring or Ethernet communications line, the content of this parameter will always be 1.

For links using an X.25 communications line, the content of this parameter is only valid when the operation parameter is X'0001', X'B001', X'B101', X'B301', or X'BF01'. It will contain the UCEP ID that was provided in the new user connection end point ID parameter on the call to the QOLSEND API with operation X'B000' or X'B400'.

Note: If an incoming X.25 SVC call is rejected by the user-defined communications application program by calling the QOLSEND API with operation X'B100', the content of this parameter will be set to zero when notification of the completion of the X'B100' operation is received from the QOLRECV API (operation X'B101').

New provider connection end point ID
OUTPUT; BINARY(4)
The provider connection end point (PCEP) ID for the connection that is to be established. This identifier must be used on all subsequent calls to the QOLSEND API for this connection.

The content of this parameter is only valid for links using an X.25 communications line and when the operation parameter is X'B201'.

Operation
OUTPUT; CHAR(2)
The type of data received by the application program. With the exception of X'0001', all values are only valid for links using an X.25 communications line. The valid values are as follows:

X'0001'	User data.
X'B001'	Completion of the X'B000' output operation.
X'B101'	Completion of the X'B100' output operation.
X'B111'	Completion of the X'B110' output operation. Cleanup of all connections complete. No data is associated with this operation.
X'B201'	Incoming X.25 switched virtual circuit (SVC) call.
X'B301'	Connection failure or reset indication received.
X'B311'	Connection failure applying to all connections for this link. This operation is only received when the extended operations parameter for the QOLELINK API is set to operations supported.
X'BF01'	Completion of the reset (X'BF00') output operation.

Note: The special value of X'0000' will be returned in the operation parameter to indicate no data was received from the QOLRECV API. See "Return and Reason Codes" on page 6-23 for more information.

Number of data units
OUTPUT; BINARY(4)
The number of data units in the input buffer that contain data. Any value between 1 and the number of data units created in the input buffer may be returned when the operation parameter is X'0001'. Otherwise, any value between 0 and 1 may be returned.

Note: The number of data units created in the input buffer was returned in the data units created parameter on the call to the QOLELINK API. See "Enable Link (QOLELINK) API" on page 6-2 for more information.

Data available
OUTPUT; CHAR(1)
Specifies whether more data is available for the user-defined communications application program to receive. The valid values are as follows:

X'00' No more data is available for the user-defined communications application program to receive.
X'01' More data is available for the user-defined communications application program to receive. The QOLRECV API should be called again.

Note: An incoming-data entry will be sent to the data queue or user queue only when the content of this parameter is X'00' and then more data is subsequently available to be received. See "Incoming-Data Entry" on page 5-3 for more information.

Diagnostic data
OUTPUT; CHAR(40)
Specifies additional diagnostic data. See "Format of Diagnostic Data Parameter" for more information.

The content of this parameter is only valid when the operation parameter is X'B001', X'B101', X'B301', X'B311', or X'BF01'.

Communications handle
INPUT; CHAR(10)
The name of the link on which to receive the data.

Format of Diagnostic Data Parameter

The format of the diagnostic data parameter is shown below. The contents of the fields within this parameter are only valid on X'B001', X'B101', X'B301', X'B311', and X'BF01' operations for the indicated return and reason codes.

Figure 6-10 (Page 1 of 2). Diagnostic Data Parameter

Field	Type	Description
Reserved	CHAR(2)	Reserved for extension.
Error code	CHAR(4)	Specifies hexadecimal diagnostic information that can be used to determine recovery actions. See "Error Codes" on page 7-9 for more information. The content of this field is only valid for 83/4001 and 83/4002 return/reason codes.
Time stamp	CHAR(8)	The time the error occurred. The content of this field is only valid for 83/4001 and 83/4002 return/reason codes.
Error log identifier	CHAR(4)	The hexadecimal identifier that can be used for locating error information in the error log. The content of this field is only valid for 83/4001 and 83/4002 return/reason codes.
Reserved	CHAR(10)	Reserved for extension.
Indicators	CHAR(1)	The indicators that the user-defined communications application program can use to diagnose a potential error condition. This is a bit-sensitive field. The valid values for bit 0 (leftmost bit) are as follows: **'0'B** Either there is no message in the QSYSOPR message queue, or there is a message and it does not have the capability to run problem analysis report (PAR) to determine the cause of the error. **'1'B** There is a message in the QSYSOPR message queue for this error, and it does have the capability to run problem analysis report (PAR) to determine the cause of the error. The valid values for bit 1 are as follows: **'0'B** The line error can be retried. **'1'B** The line error is not able to be restarted. The valid values for bit 2 are as follows: **'0'B** The cause and diagnostic codes fields are not valid. **'1'B** The cause and diagnostic codes fields are valid. The valid values for bit 3 are as follows: **'0'B** The error has not been reported to the system operator message queue. **'1'B** The error has been reported to the system operator message queue. The valid values for bit 4 are as follows: **'0'B** A reset request packet was transmitted on the network **'1'B** A reset confirmation packet was transmitted on the network instead of a reset request packet. The content of bit 4 is only valid for operation X'BF01' with 00/0000 return/reason codes. The content of the indicators field is only valid for 83/4001, 83/4002, and 83/3202 return/reason codes, and 00/0000 return/reason codes for operation X'BF01'.

Figure 6-10 (Page 2 of 2). Diagnostic Data Parameter

Field	Type	Description
X.25 cause code	CHAR(1)	Specifies additional information on the condition reported. See the *X.25 Network Guide* for interpreting the values of this field. The content of this field is only valid for 83/4001, 83/4002 and 83/3202 return/reason codes.
X.25 diagnostic code	CHAR(1)	Specifies additional information on the condition reported. See the *X.25 Network Guide* for interpreting the values of this field. The content of this field is only valid for 83/4001, 83/4002 and 83/3202 return/reason codes.
Reserved	CHAR(1)	Reserved for extension.
Error offset	BINARY(4)	The offset from the top of the input buffer to the incorrect data in the input buffer. The content of this field is only valid for a 83/1999 return/reason code.
Reserved	CHAR(4)	Reserved for extension.

LAN Input Operations

The only type of data that an application program can receive from the QOLRECV API on links using a token-ring or Ethernet communications line is user data (operation X'0001'). User-defined communications support returns the following information for each data frame received from the QOLRECV API:

- One or more data units. The first data unit contains a general LAN header, routing information if a token ring is used, and user data.
- Total length of the data unit. This information is reported in the corresponding input buffer descriptor element.

For example, suppose two data frames came in from the network and the user-defined communications application program was notified of this by an incoming-data entry on the data queue or user queue. On return from the QOLRECV API, the information for the first frame would be in the first data unit of the input buffer and described in the first element of the input buffer descriptor. The information for the second frame would be in the second data unit of the input buffer and described in the second element of the input buffer descriptor. The number of data units parameter would be set to 2.

Data Unit Format—LAN Operation X'0001': Each data frame received from the QOLRECV API corresponds to a data unit in the input buffer. The information in each of these data units is made up of a general LAN header, routing information (for token-ring links only), followed by user data.

The general LAN header is used to pass information about the frame to the communications support. The fields in the general LAN header are used for all LAN link types, although some of them are link specific. For example, routing information is only for token-ring links, and the length of routing information is X'00' to X'18'. For non-token-ring links, the length of the routing information is always X'00'. Also, DSAP and SSAP are defined for protocols that use the 802.2 logical link control interface and do not apply to Ethernet Version 2. A DSAP and SSAP of X'00' tells the communications support that the data frame is an Ethernet Version 2 frame.

The general LAN header is described in Figure 6-11.

Figure 6-11 (Page 1 of 2). Format of the General LAN Information

Field	Type	Description
Length of general LAN information	BINARY(2)	The length of the general LAN information in the data unit, including this field. This field is always set to 16.
Sending adapter address	CHAR(6)	Specifies, in packed form, the adapter address from which this frame was sent. The possible values returned in this field depend on the filters activated for this link. See "Set Filter (QOLSETF) API" on page 6-43 for more information. **Note**: Because user-defined communications support only allows connectionless service over LANs, all frames received on a single call to the QOLRECV API may not have the same source adapter address.
DSAP address	CHAR(1)	The service access point on which the AS/400 system received this frame. The possible values returned in this field depend on the filters activated for this link. See "Set Filter (QOLSETF) API" on page 6-43 for more information. **Note**: The Ethernet Version 2 standard does not define a DSAP address in an Ethernet Version 2 frame. Therefore, when receiving Ethernet Version 2 frames, the DSAP address will be null (X'00').

Figure 6-11 (Page 2 of 2). Format of the General LAN Information

Field	Type	Description
SSAP address	CHAR(1)	The service access point on which the source system sent this frame. The possible values returned in this field depend on the filters activated for this link. See "Set Filter (QOLSETF) API" on page 6-43 for more information. **Note**: The Ethernet Version 2 standard does not define a SSAP address in an Ethernet Version 2 frame. Therefore, when receiving Ethernet Version 2 frames, the SSAP address will be null (X'00').
Reserved	CHAR(2)	Reserved for extension.
Length of token-ring routing information	BINARY(2)	The length of the routing information in the data unit. For links using a token-ring communications line, any value between 0 and 18 may be returned, where 0 indicates that there is no routing information. For links using an Ethernet communications line, the content of this field is not applicable and will be set to 0 indicating that there is no routing information.
Length of user data	BINARY(2)	The length of the user data in the data unit. This will be less than or equal to the maximum frame size allowed on the service access point returned in the DSAP address field. See "Query Line Description (QOLQLIND) API" on page 6-5 to determine the maximum frame size allowed on the service access point returned in the DSAP address field. For Ethernet Version 2 frames, this will be at least 48 and not more than 1502 (including 2 bytes for the Ethernet type field). **Note**: Ethernet 802.3 frames will be padded when the user data is less than 46 bytes.

Token-ring routing information follows the general LAN header. The length of this field is specified by the length of token-ring routing information field found in the general LAN header. If the length of the routing information is nonzero, the user data follows the routing information header.

Figure 6-12 shows the fields and offsets used for Ethernet 802.3 and token-ring frames without routing information.

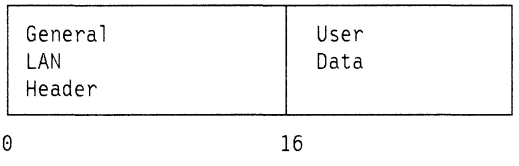


Figure 6-12. Ethernet 802.3 and Token-Ring Frames with No Routing Information

The length of the user data is described in the length of user data field in the general LAN header. For Ethernet Version 2 frames, the first 2 bytes of user data are used for the *frame type*. The type field is a 2-byte field that specifies the upper layer protocol of the frame.

The adapter address, DSAP, SSAP, and frame type fields are all used to define inbound routing information used by the QOLSETF API. Refer to "Set Filter (QOLSETF) API" on page 6-43 for information on the QOLSETF API and how inbound routing information is used to route inbound data to the application program.

Note: Inbound routing information is not related to the token-ring routing information described in the general LAN header.

Figure 6-13 shows the fields and offsets used for token-ring frames with routing information.

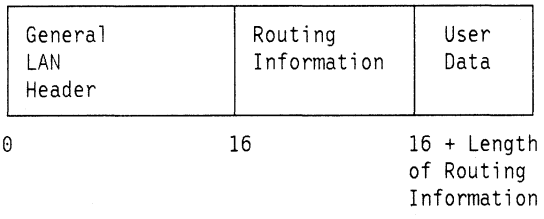


Figure 6-13. Token-Ring Frames with Routing Information

Figure 6-14 shows the fields and offsets used for Ethernet Version 2 frames.

Note: For Ethernet Version 2, the frame type field is the first 2 bytes of user data, following the general LAN information, with user data starting at offset 18.

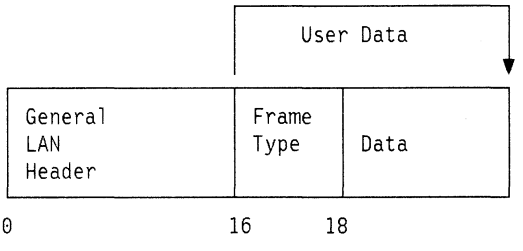


Figure 6-14. Ethernet Version 2 Frames

Input Buffer Descriptor Element Format—LAN Operation X'0001': The information returned in each data unit of the input buffer will be described in the corresponding element of the input buffer descriptor.

Figure 6-15 shows the format of each element in the input buffer descriptor.

Figure 6-15. Format of an Element in the Input Buffer Descriptor

Field	Type	Description
Length	BINARY(2)	The number of bytes of information in the corresponding data unit of the input buffer. This will be equal to the length of the general LAN information with the length of the routing information and the length of the user data. See Figure 6-11 for general LAN information fields and descriptions.
Reserved	CHAR(30)	Reserved for extension.

X.25 SVC and PVC Input Operations

Figure 6-16 shows the types of data that can be received from the QOLRECV API on links using an X.25 communications line.

Figure 6-16. X.25 SVC and PVC Input Operations

Operation	Meaning
X'0001'	User data (SVC or PVC).
X'B001'	Completion of the X'B000' output operation (SVC or PVC).
X'B101'	Completion of the X'B100' output operation (SVC or PVC).
X'B201'	Incoming X.25 call (SVC).
X'B301'	Connection failure or reset indication (SVC or PVC).
X'B311'	Connection failure applying to all connections for this link.
X'BF01'	Completion of the X'BF00' output operation (SVC or PVC).

X.25 Operation X'0001': This operation indicates that user data was received on an X.25 SVC or PVC connection. User-defined communications support will return the following information:

- User data in the next data unit of the input buffer, starting with the first data unit
- A description, in the corresponding element of the input buffer descriptor, of the user data in that data unit

For example, suppose two data units of user data came in from the network and the application program was notified of this by an incoming-data entry on the data queue or user queue. On return from the QOLRECV API, the first portion of the user data would be in the first data unit of the input buffer and described in the first element of the input buffer descriptor. The second portion of the user data would be in the second data unit of the input buffer and described in the second element of the input buffer descriptor. The number of data units parameter would be set to 2.

User-defined communications support will automatically reassemble the X.25 data packet(s) from a complete packet sequence into the next data unit of the input buffer. If the amount of user data in a complete packet sequence is more than what can fit into a data unit, the more data indicator field in the corresponding element of the input buffer descriptor will be set to X'01' and the next data unit will be used for the remaining user data, and so on.

Data Unit Format—X.25 Operation X'0001': Each data unit in the input buffer consists solely of user data and starts offset 0 from the top of the data unit.

Input Buffer Descriptor Element Format—X.25 Operation X'0001': The user data returned in each data unit of the input buffer will be described in the corresponding element of the input buffer descriptor.

Figure 6-17 shows the format of each element in the input buffer descriptor.

Figure 6-17 (Page 1 of 2). Format of an Element in the Input Buffer Descriptor

Field	Type	Description
Length	BINARY(2)	The number of bytes of user data in the corresponding data unit of the input buffer. This will always be less than or equal to the X.25 user data size parameter that was specified on the call to the QOLELINK API when the link was enabled. See "Enable Link (QOLELINK) API" on page 6-2 for more information. **Note**: The maximum amount of user data in a data unit of the input buffer may be further limited by the maximum data unit assembly size for a connection. See "Send Data (QOLSEND) API" on page 6-27 for more information.
More data indicator	CHAR(1)	Specifies whether the remaining amount of user data from a complete X.25 packet sequence is more than can fit into the corresponding data unit. The valid values are as follows: **X'00'** The remaining amount of user data from a complete X.25 packet sequence fit into the corresponding data unit. **X'01'** The remaining amount of user data from a complete X.25 packet sequence could not all fit into the corresponding data unit. The next data unit will be used.

Figure 6-17 (Page 2 of 2). Format of an Element in the Input Buffer Descriptor

Field	Type	Description
Qualified data indicator	CHAR(1)	Specifies whether the X.25 qualifier bit (Q-bit) was set on or off in all X.25 packets reassembled into the corresponding data unit. The valid values are as follows: **X'00'** The Q-bit was set off in all X.25 packets reassembled into the corresponding data unit. **X'01'** The Q-bit was set on in all X.25 packets reassembled into the corresponding data unit.
Interrupt packet indicator	CHAR(1)	Specifies whether the user data in the corresponding data unit was received in an X.25 interrupt packet. The valid values are as follows: **X'00'** The user data in the corresponding data unit was received in one or more data packets. **X'01'** The user data in the corresponding data unit was received in an X.25 interrupt packet.
Delivery confirmation indicator	CHAR(1)	Specifies whether the X.25 delivery confirmation bit (D-bit) was set on or off in all X.25 packets reassembled into the corresponding data unit. The valid values are as follows: **X'00'** The D-bit was set off in all X.25 packets reassembled into the corresponding data unit. **X'01'** The D-bit was set on in all X.25 packets reassembled into the corresponding data unit. **Note**: A packet-level confirmation is sent by the input/output processor (IOP) when a packet is received with the X.25 D-bit set on.
Reserved	CHAR(26)	Reserved for extension.

X.25 Operation X'B001': This operation indicates that a X'B000' output operation has completed. User-defined communications support will return the data for this operation (if any) in the first data unit of the input buffer. The input buffer descriptor is not used.

Data will be returned in the input buffer for the following return and reason codes:

- 0/0
- 83/1999
- 83/4002 (only when the number of data units parameter is set to one)

The format of the data returned in the input buffer for the X'B001' operation depends on whether the X'B000' output operation was used to initiate an SVC call or to open a PVC connection. Each format will be explained below.

Note: The formats below only apply to 0/0 and 83/4002 return and reason codes. When the X'B001' operation is received with a 83/1999 return and reason code, the data returned starts at offset 0 from the top of the first data unit in the input buffer and contains the data specified in the output buffer on the X'B000' output operation. See "Send Data (QOLSEND) API" on page 6-27 for more information.

Data Unit Format—X.25 Operation X'B001' (Completion of SVC Call): The data returned starts at offset 0 from the top of the first data unit in the input buffer.

Figure 6-18 shows the format of the data returned for the X'B001' operation.

Figure 6-18 (Page 1 of 2). Format of Data for X'B001' Operation (Completion of SVC Call)

Field	Type	Description
Reserved	CHAR(2)	Reserved for extension.
Logical channel identifier	CHAR(2)	The logical channel identifier assigned to the SVC connection.[1]
Transmit packet size	BINARY(2)	The negotiated transmit packet size for this connection.[1]
Transmit window size	BINARY(2)	The negotiated transmit window size for this connection.[1]
Receive packet size	BINARY(2)	The negotiated receive packet size for this connection.[1]
Receive window size	BINARY(2)	The negotiated receive window size for this connection.[1]
Reserved	CHAR(32)	Reserved for extension.

Figure 6-18 (Page 2 of 2). Format of Data for X'B001' Operation (Completion of SVC Call)

Field	Type	Description
Delivery confirmation support	CHAR(1)	Specifies whether the X.25 delivery confirmation bit (D-bit) was set on or off in the call connected packet. This also specifies the D-bit support for this connection.[1] The valid values are as follows: **X'00'** The D-bit was set off in the call connected packet. D-bit will be supported for sending data but not for receiving data. **Note**: When this value is returned and an X.25 packet is received with the D-bit set on, the input/output processor (IOP) will send a reset packet. **X'01'** The D-bit was set on in the call connected packet. D-bit will be supported for sending data and for receiving data.
Reserved	CHAR(11)	Reserved for extension.
X.25 facilities length	BINARY(1)	The number of bytes of data in the X.25 facilities field. Any value between 0 and 109 may be returned.
X.25 facilities	CHAR(109)	The X.25 facilities data.
Reserved	CHAR(48)	Reserved for extension.
Call/clear user data length	BINARY(2)	The number of bytes of data in the call/clear user data field. Any value between 0 and 128 may be returned.
Call/clear user data	CHAR(128)	For a 0/0 return and reason code, this specifies the call user data. For an 83/4002 return and reason code, this specifies the clear user data.
Reserved	CHAR(168)	Reserved for extension.

1 The content of this field is only valid for a 0/0 return and reason code.

Data Unit Format—X.25 Operation X'B001' (Completion of Open PVC): The data returned starts at offset 0 from the top of the first data unit in the input buffer.

Figure 6-19 shows the format of the data returned for the X'B001' operation.

Figure 6-19. Format of Data for X'B001' Operation (Completion of Open PVC)

Field	Type	Description
Reserved	CHAR(4)	Reserved for extension.
Transmit packet size	BINARY(2)	The negotiated transmit packet size for this connection. **Note**: This will be the same as the requested transmit packet size specified on the X'B000' output operation.
Transmit window size	BINARY(2)	The negotiated transmit window size for this connection. **Note**: This will be the same as the requested transmit window size specified on the X'B000' output operation.
Receive packet size	BINARY(2)	The negotiated receive packet size for this connection. **Note**: This will be the same as the requested receive packet size specified on the X'B000' output operation.
Receive window size	BINARY(2)	The negotiated receive window size for this connection. **Note**: This will be the same as the requested receive window size specified on the X'B000' output operation.
Reserved	CHAR(500)	Reserved for extension.

X.25 Operation X'B101': This operation indicates that a X'B100' output operation has completed. User-defined communications support will return the data for this operation (if any) in the first data unit of the input buffer. The input buffer descriptor is not used.

Data will be returned in the input buffer for the following return and reason codes:

- 0/0 (only when the number of data units parameter is set to one)
- 83/1999

Note: The format below only applies for a 0/0 return and reason code. When the X'B101' operation is received with an 83/1999 return and reason code, the data returned starts at offset 0 from the top of the first data unit in the input buffer and contains the data specified in the output buffer on the X'B100' output operation. See "Send Data (QOLSEND) API" on page 6-27 for more information.

Data Unit Format—X.25 Operation X'B101': The data returned starts at offset 0 from the top of the first data unit in the input buffer.

Figure 6-20 shows the format of the data returned for the X'B101' operation.

Figure 6-20. Format of Data for X'B101' Operation

Field	Type	Description
Clear type	CHAR(2)	The type of clear user data returned. The valid values are as follows: **X'0001'** Clear confirmation data included. **X'0002'** Clear indication data included.
Cause code	CHAR(1)	The X.25 cause code.
Diagnostic code	CHAR(1)	The X.25 diagnostic code.
Reserved	CHAR(4)	Reserved for extension.
X.25 facilities length	BINARY(1)	The number of bytes of data in the X.25 facilities field. Any value between 0 and 109 may be returned.
X.25 facilities	CHAR(109)	The X.25 facilities data.
Reserved	CHAR(48)	Reserved for extension.
Clear user data length	BINARY(2)	The number of bytes of data in the clear user data field. Any value between 0 and 128 may be returned.
Clear user data	CHAR(128)	The clear user data.
Reserved	CHAR(216)	Reserved for extension.

X.25 Operation X'B111': This operation indicates a X'B110' output operation has completed. All connections have been closed and the clean up of connection control information is complete. All UCEPs and PCEPs are freed. There is no data associated with this operation.

X.25 Operation X'B201': This operation indicates that an incoming X.25 SVC call was received. User-defined communications support returns the data for this operation in the first data unit of the input buffer. The input buffer descriptor is not used.

Note: It is the responsibility of the application program to either accept or reject the incoming call. This is done by calling the QOLSEND API with operation X'B400' or X'B100', respectively.

Data Unit Format—X.25 Operation X'B201': The data returned starts at offset 0 from the top of the first data unit in the input buffer.

Figure 6-21 shows the format of the data returned for the X'B201' operation.

Figure 6-21 (Page 1 of 2). Format of Data for X'B201' Operation

Field	Type	Description
Reserved	CHAR(2)	Reserved for extension.
Logical channel identifier	CHAR(2)	The logical channel identifier assigned to the incoming SVC call.
Transmit packet size	BINARY(2)	The requested transmit packet size for this connection.
Transmit window size	BINARY(2)	The requested transmit window size for this connection.
Receive packet size	BINARY(2)	The requested receive packet size for this connection.
Receive window size	BINARY(2)	The requested receive window size for this connection.
Reserved	CHAR(7)	Reserved for extension.
Calling DTE address length	BINARY(1)	The number of binary coded decimal (BCD) digits in the calling DTE address.

Figure 6-21 (Page 2 of 2). Format of Data for X'B201' Operation

Field	Type	Description
Calling DTE address	CHAR(16)	Specifies, in binary coded decimal (BCD), the calling DTE address. The address will be left justified and padded on the right with BCD zeros.
Reserved	CHAR(8)	Reserved for extension.
Delivery confirmation support	CHAR(1)	Specifies whether the X.25 delivery confirmation bit (D-bit) was set on or off in the incoming call packet. The valid values are as follows: **X'00'** The D-bit was set off in the incoming call packet. **X'01'** The D-bit was set on in the incoming call packet.
Reserved	CHAR(9)	Reserved for extension.
Reverse charging indicator	CHAR(1)	Specifies reverse charging options. The valid values are as follows: **X'00'** Reverse charging not requested. **X'01'** Reverse charging requested.
Fast select indicator	CHAR(1)	Specifies fast select options. The valid values are as follows: **X'00'** Fast select not requested. **X'01'** Fast select with restriction requested. **X'02'** Fast select without restriction requested.
X.25 facilities length	BINARY(1)	The number of bytes of data in the X.25 facilities field. Any value between 0 and 109 may be returned.
X.25 facilities	CHAR(109)	The X.25 facilities data.
Reserved	CHAR(48)	Reserved for extension.
Call user data length	BINARY(2)	The number of bytes of data in the call user data field. Any value between 0 and 128 may be returned.
Call user data	CHAR(128)	The call user data. **Note**: The AS/400 system treats the first byte of call user data as the protocol identifier (PID).
Reserved	CHAR(168)	Reserved for extension.

X.25 Operation X'B301': This operation indicates that a failure has occurred, or a reset indication has been received, on an X.25 SVC or PVC connection. User-defined communications support will return data for this operation in the first data unit of the input buffer only on a 83/4002 return and reason code when the number of data units parameter is set to one. The input buffer descriptor is not used.

Note: The diagnostic data parameter will contain the X.25 cause and diagnostic codes when a reset indication is received.

Data Unit Format—X.25 Operation X'B301': The data returned starts at offset 0 from the top of the first data unit in the input buffer.

Figure 6-22 shows the format of the data returned for the X'B301' operation.

Figure 6-22. Format of Data for X'B301' Operation

Field	Type	Description
Reserved	CHAR(8)	Reserved for extension.
X.25 facilities length	BINARY(1)	The number of bytes of data in the X.25 facilities field. Any value between 0 and 109 may be returned.
X.25 facilities	CHAR(109)	The X.25 facilities data.
Reserved	CHAR(48)	Reserved for extension.
Clear user data length	BINARY(2)	The number of bytes of data in the clear user data field. Any value between 0 and 128 may be returned.
Clear user data	CHAR(128)	The clear user data.
Reserved	CHAR(216)	Reserved for extension.

X.25 Operation X'B311': This operation indicates that an error has occurred that has caused the system to close all connections on the link. The error may be a system error or a network error. The error information is returned in the diagnostic data and no additional data is provided.

Note: This operation is only received when the extended operation parameter on the QOLELINK API is set to operation supported. If the extended operations are not supported and an error occurs that will close all connections, X'B301' is received for each connection.

X.25 Operation X'BF01': This operation indicates that a X'BF00' output operation has been completed. Neither the input buffer nor the input buffer descriptor is used for this operation.

Note: When the X'BF01' operation is received with a 0/0 return and reason code, the diagnostic data parameter will contain information indicating if a reset request or reset confirmation packet was sent.

Return and Reason Codes

The return and reason codes that can be returned from the QOLRECV API depend on the type of communications line the link is using and on the type of data (operation) that was received.

LAN Return and Reason Codes

Return and Reason Codes Indicating No Data Received: Figure 6-23 shows the return and reason codes that indicate data could not be received from the QOLRECV API.

Note: When these return and reason codes are returned, all output parameters except the return and reason codes will contain hexadecimal zeros.

Figure 6-23. Return and Reason Codes Indicating No Data Received

Return / Reason Code	Meaning	Recovery
0/3203	No data available to be received.	Ensure that user-defined communications support has data available to be received before calling the QOLRECV API. Try the request again.
80/2200	Queue error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Correct the error, enable the link, and try the request again.
80/2401	Input buffer or input buffer descriptor error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Correct the error, enable the link, and try the request again.
80/3002	A previous error occurred on this link that was reported to the application program by escape message CPF91F0 or CPF91F1. However, the application program has attempted another operation.	Ensure the link is disabled and see messages in the job log for further information. If escape message CPF91F0 was sent to the application program, then report the problem using the ANZPRB command. Otherwise, correct the error, enable the link, and try the request again.
80/4000	Error recovery has been canceled for this link.	Ensure the link is disabled and see messages in the job log for further information. Correct the condition, enable the link, and try the request again.
80/9999	Internal system error detected. Escape message CPF91F0 will be sent to the application program when this return and reason code is received.	See messages in the job log for further information. Report the problem using the ANZPRB command.
83/3001	Link not enabled.	Correct the communications handle parameter. Try the request again.
83/3004	Link is enabling.	Wait for the enable-complete entry to be sent to the data queue or user queue. If the link was successfully enabled, try the request again.

Return and Reason Codes for LAN Operation X'0001'

Figure 6-24. Return and Reason Codes for LAN Operation X'0001'.

Return / Reason Code	Meaning	Recovery
0/0	User data received successfully.	Continue processing.

X.25 Return and Reason Codes

Return and Reason Codes Indicating No Data Received:
Figure 6-25 shows the return and reason codes that indicate data could not be received from the QOLRECV API.

Note: When these return and reason codes are returned, all output parameters except the return and reason codes will contain hexadecimal zeros.

Figure 6-25. Return and Reason Codes Indicating No Data Received

Return / Reason Code	Meaning	Recovery
0/3203	No data available to be received.	Ensure that user-defined communications support has data available to be received before calling the QOLRECV API. Try the request again.
80/2200	Queue error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Correct the error, enable the link, and try the request again.
80/2401	Input buffer or input buffer descriptor error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Correct the error, enable the link, and try the request again.
80/3002	A previous error occurred on this link that was reported to the application program by escape message CPF91F0 or CPF91F1. However, the application program has attempted another operation.	Ensure the link is disabled and see messages in the job log for further information. If escape message CPF91F0 was sent to the application program, then report the problem using the ANZPRB command. Otherwise, correct the error, enable the link, and try the request again.
80/4000	Error recovery has been canceled for this link.	Ensure the link is disabled and see messages in the job log for further information. Correct the condition, enable the link, and try the request again.
80/9999	Internal system error detected. Escape message CPF91F0 will be sent to the application program when this return and reason code is received.	See messages in the job log for further information. Report the problem using the ANZPRB command.
83/3001	Link not enabled.	Correct the communications handle parameter. Try the request again.
83/3004	Link is enabling.	Wait for the enable-complete entry to be sent to the data queue or user queue. If the link was successfully enabled, try the request again.

Return and Reason Codes for X.25 Operation X'0001'

Figure 6-26. Return and Reason Codes for X.25 Operation X'0001'

Return / Reason Code	Meaning	Recovery
0/0	User data received successfully.	Continue processing.

Return and Reason Codes for X.25 Operation X'B001'

Figure 6-27. Return and Reason Codes for X.25 Operation X'B001'

Return / Reason Code	Meaning	Recovery
0/0	The X'B000' output operation was successful.	Continue processing.
83/1999	Incorrect data was specified in output buffer when the X'B000' output operation was issued. **Note**: The data specified in the output buffer will be copied into the input buffer and the error offset field in the diagnostic data parameter will point to the incorrect data.	Correct the incorrect data. Then, try the X'B000' output operation again.
83/3204	Connection ending because a X'B100' output operation was issued.	Wait for notification of the completion of the X'B100' output operation from the QOLRECV API (X'B101' operation).
83/4001	Link failure, system starting error recovery for this link. The connection has ended.	Wait for the link to recover. Then, try the X'B000' output operation again.
83/4002	Connection failure. The connection has ended. The diagnostic data parameter will contain more information on this error.	Correct any errors and try the X'B000' output operation again.
83/4005	All SVC channels are currently in use, or the requested PVC channel is already in use.	Wait for a virtual circuit to become available. Then, try the X'B000' output operation again.

Return and Reason Codes for X.25 Operation X'B101'

Figure 6-28. Return and Reason Codes for X.25 Operation X'B101'

Return / Reason Code	Meaning	Recovery
0/0	The X'B100' output operation was successful. The connection has ended.	Continue processing.
83/1007	Connection identifier not valid because connection has already ended.	Continue processing.
83/1999	Incorrect data was specified in output buffer when the X'B100' output operation was issued. **Note**: The data specified in the output buffer will be copied into the input buffer and the error offset field in the diagnostic data parameter will point to the incorrect data.	Correct the incorrect data. Then, try the X'B100' output operation again.

Return and Reason Codes for X.25 Operation X'B111'

Figure 6-29. Return and Reason Codes for X.25 Operation X'B111'

Return / Reason Code	Meaning	Recovery
0/0	The X'B100' output operation was successful. The connection has ended.	Continue processing.
83/1007	Connection identifier not valid because connection has already ended.	Continue processing.
83/3205	The X'B110' operation is rejected because the application has not received the X'B311' operation prior to requesting the X'B110' operation.	Correct the application.

Return and Reason Codes for X.25 Operation X'B201'

Figure 6-30. Return and Reason Codes for X.25 Operation X'B201'

Return / Reason Code	Meaning	Recovery
0/0	Incoming X.25 SVC call received successfully.	Continue processing.

Return and Reason Codes for X.25 Operation X'B301'

Figure 6-31. Return and Reason Codes for X.25 Operation X'B301'

Return / Reason Code	Meaning	Recovery
83/3201	The maximum amount of incoming user data that can be held by user-defined communications support for the application program on this connection has been exceeded.	Issue the X'B100' output operation to end the connection.
83/3202	A reset indication has been received on this connection. The X.25 cause and diagnostic code fields in the diagnostic data parameter will contain the cause and diagnostic codes of the reset indication.	Issue the X'BF00' output operation to send a reset confirmation packet.
83/4001	Link failure, system starting error recovery for this link.	Issue the X'B100' output operation to end the connection.
83/4002	Connection failure. The diagnostic data parameter will contain more information on this error.	Issue the X'B100' output operation to end the connection.

Return and Reason Codes for X.25 Operation X'B311'

Figure 6-32. Return and Reason Codes for X.25 Operation X'B311'

Return / Reason Code	Meaning	Recovery
83/4001	Link failure, system starting error recovery for this link. All connections that were active on this link are closed or cleared.	Issue the X'B110' operation to free the connections.
83/4002	A network error has occurred that affects all connections on this link. All connections that were active on this link are closed or cleared. The diagnostic data contains more information on this error.	Issue the X'B110' operation to free the connections.

Return and Reason Codes for X.25 Operation X'BF01'

Figure 6-33 (Page 1 of 2). Return and Reason Codes for X.25 Operation X'BF01'

Return / Reason Code	Meaning	Recovery
0/0	The X'BF00' output operation was successful. The diagnostic data parameter will contain information indicating if a reset request or reset confirmation packet was sent.	Continue processing.
83/1006	Operation not valid.	Do not issue the X'BF00' output operation on connections that do not support resets.
83/3201	The maximum amount of incoming user data that can be held by user-defined communications support for the application program on this connection has been exceeded.	Wait to receive a failure notification from the QOLRECV API indicating this condition (X'B301' operation, 83/3201 return and reason code). Then issue the X'B100' output operation to end the connection.
83/3204	Connection ending because a X'B100' output operation was issued.	Wait for notification of the completion of the X'B100' output operation from the QOLRECV API (X'B101' operation).

Figure 6-33 (Page 2 of 2). Return and Reason Codes for X.25 Operation X'BF01'

Return / Reason Code	Meaning	Recovery
83/4001	Link failure, system starting error recovery for this link.	Wait to receive a failure notification from the QOLRECV API indicating this condition (X'B301' operation, 83/4001 return and reason code). Then, issue the X'B100' output operation to end the connection.
83/4002	Connection failure.	Wait to receive a failure notification from the QOLRECV API indicating this condition (X'B301' operation, 83/4002 return and reason code). Then, issue the X'B100' output operation to end the connection.

Error Messages

CPF91F0 E Internal system error.
CPF91F1 E User-defined communications application error.

Send Data (QOLSEND) API

Parameters

Required Parameter Group:

1	Return code	Output	Binary(4)
2	Reason code	Output	Binary(4)
3	Diagnostic data	Output	Char(40)
4	New provider connection end point ID	Output	Binary(4)
5	New user end point connection ID	Input	Binary(4)
6	Existing provider connection end point ID	Input	Binary(4)
7	Communications handle	Input	Char(10)
8	Operation	Input	Char(2)
9	Number of data units	Input	Binary(4)

The Send Data (QOLSEND) API performs output on a link that is currently enabled in the job in which the application program is running. The operation parameter allows you to specify the type of output operation to perform. The application program must provide the data associated with the output operation in the output buffer that was created when the link was enabled. For X'0000' operations, the application program must also provide a description of that data in the output buffer descriptor that was created when the link was enabled.

The types of output operations that can be performed on a link depend on the type of communications line that the link is using. See "LAN Output Operations" on page 6-29 for more information on output operations that are supported on links using a token-ring or Ethernet communications line. See "X.25 SVC and PVC Output Operations" on page 6-31 for more information on output operations that are supported on links using an X.25 communications line.

Required Parameter Group

Return code
OUTPUT; BINARY(4)
The recovery action to take. See "Return and Reason Codes" on page 6-39.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-39.

Diagnostic data
OUTPUT; CHAR(40)
Additional diagnostic data. See "Diagnostic Data Parameter Format" on page 6-28 for more information.

The content of this parameter is only valid when the operation parameter is set to X'0000' or X'B400'.

New provider connection end point ID
OUTPUT; BINARY(4)
The provider connection end point (PCEP) ID for the connection that is to be established. This identifier must be used on all subsequent calls to the QOLSEND API for this connection.

The content of this parameter is only valid for links using an X.25 communications line and when the operation parameter is set to X'B000'.

New user connection end point ID
INPUT; BINARY(4)
The user connection end point (UCEP) ID for the connection that is to be established. This is the identifier on which all incoming data for this connection will be received. Any numeric value except zero should be used. See "Receive Data (QOLRECV) API" on page 6-14 for more information.

The content of this parameter is only valid for links using an X.25 communications line and when the operation parameter is set to X'B000' or X'B400'.

Existing provider connection end point ID
INPUT; BINARY(4)
The PCEP ID for the connection on which this operation will be performed. For links using a token-ring or Ethernet communications line, the content of this parameter must always be set to 1.

For links using an X.25 communications line, the content of this parameter is only valid when the operation parameter is set to X'0000', X'B100', X'B400', or

X'BF00'. It must contain the PCEP ID that was returned in the new provider connection end point ID parameter from the call to the QOLSEND API with operation X'B000', or the PCEP ID that was returned in the new provider connection end point ID parameter from the call to the QOLRECV API with operation X'B201' (incoming call). See "Receive Data (QOLRECV) API" on page 6-14 for more information on receiving X.25 calls.

Communications handle
INPUT; CHAR(10)
The name of the link on which to perform the output operation.

Operation
INPUT; CHAR(2)
The type of output operation to perform. With the exception of X'0000', all values are only valid for links using an X.25 communications line. The valid values are as follows:

X'0000'	Send data.
X'B000'	Send call request packet (SVC) or open PVC connection.
X'B100'	Send clear packet (SVC) or close PVC connection.
X'B110'	Initiate final cleanup of all connections that were closed by the system. This operation is only valid when the application receives an X'B311' operation to receive connection failure data.
X'B400'	Send call accept packet (SVC).
X'BF00'	Send reset request packet or reset confirmation packet (SVC or PVC).

Number of data units
INPUT; BINARY(4)
The number of data units in the output buffer that contain data. Any value between 1 and the number of data units created in the output buffer may be used.

The content of this parameter is only valid when the operation parameter is set to X'0000'.

Note: The number of data units created in the output buffer was returned in the data units created parameter on the call to the QOLELINK API. See "Enable Link (QOLELINK) API" on page 6-2 for more information.

Diagnostic Data Parameter Format: The format of the diagnostic data parameter is shown below. The contents of the fields within this parameter are only valid on X'0000' and X'B400' operations for the indicated return and reason codes.

Figure 6-34 (Page 1 of 2). Diagnostic Data Parameter

Field	Type	Description
Reserved	CHAR(2)	Reserved for extension.
Error code	CHAR(4)	Specifies hexadecimal diagnostic information that can be used to determine recovery actions. See "Error Codes" on page 7-9 for more information. The content of this field is only valid for 83/4001, 83/4002, and 83/4003 return/reason codes.
Time stamp	CHAR(8)	The time the error occurred. The content of this field is only valid for 83/4001, 83/4002, and 83/4003 return/reason codes.
Error log identifier	CHAR(4)	The hexadecimal identifier that can be used for locating error information in the error log. The content of this field is only valid for 83/4001, 83/4002, and 83/4003 return/reason codes.
Reserved	CHAR(10)	Reserved for extension.

Figure 6-34 (Page 2 of 2). Diagnostic Data Parameter

Field	Type	Description
Indicators	CHAR(1)	Specifies indicators the user-defined communications application program can use for diagnosing a potential error condition. This is a bit sensitive field. The valid values for bit 0 (leftmost bit) are as follows: **'0'B** Either there is no message in the QSYSOPR message queue, or there is a message and it does not have the capability to run problem analysis report (PAR) to determine the cause of the error. **'1'B** There is a message in the QSYSOPR message queue for this error, and it does have the capability to run problem analysis report (PAR) to determine the cause of the error. The valid values for bit 1 are as follows: **'0'B** The line error can be retried. **'1'B** The line error cannot be retried. The valid values for bit 2 are as follows: **'0'B** The cause and diagnostic codes fields are not valid. **'1'B** The cause and diagnostic codes fields are valid. The valid values for bit 3 are as follows: **'0'B** The error has not been reported to the system operator message queue. **'1'B** The error has been reported to the system operator message queue. For example, consider the following values for the indicators field: **X'20'** A condition has caused X.25 cause and diagnostic codes to be passed to the application. This information can determine the cause of the condition. **X'50'** An error has occurred and been reported to the QSYSOPR message queue. The error cannot be retried. **X'F0'** An error has occurred and been reported to the QSYSOPR message queue. The error cannot be retried, and has X.25 cause and diagnostic codes associated with it. Also a problem analysis report can be generated to determine the probable cause. The content of this field is only valid for 83/4001, 83/4002, 83/3202 and 83/4003 return/reason codes.
X.25 cause code	CHAR(1)	Specifies additional information on the condition reported. See the *X.25 Network Guide* for interpreting the values of this field. The content of this field is only valid for 83/4001, 83/4002 and 83/3202 return/reason codes.
X.25 diagnostic code	CHAR(1)	Specifies additional information on the condition reported. See the *X.25 Network Guide* for interpreting the values of this field. The content of this field is only valid for 83/4001, 83/4002 and 83/3202 return/reason codes.
Reserved	CHAR(1)	Reserved for extension.
Error offset	BINARY(4)	The offset from the top of the output buffer to the incorrect data in the output buffer. The content of this field is only valid for a 83/1999 return/reason code.
Reserved	CHAR(4)	Reserved for extension.

LAN Output Operations

The only output operation supported on links using a token-ring or Ethernet communications line is X'0000' (send user data). For each data frame to be sent on the network, the application program must provide the following information:

- General LAN information, optional routing information, and user data in the next data unit of the output buffer, starting with the first data unit
- A description, in the corresponding element of the output buffer descriptor, of the information in that data unit.

For example, suppose a user-defined communications application program wants to send two data frames. The information for the first frame would be placed in first data unit of the output buffer and described in the first element of the output buffer descriptor. The information for the second frame would be placed in the second data unit of the output buffer and described in the second element of the output buffer descriptor. The number of data units parameter on the call to the QOLSEND API would be set to 2.

Note: The X'0000' operation is synchronous. Control will not return from the QOLSEND API until the operation completes.

Data Unit Format—LAN Operation X'0000': Each data frame to be sent on the network corresponds to a data unit in the output buffer. The information in each of these data units is made up of general LAN information, optional routing data, and user data.

Figure 6-35 shows the format of the general LAN information.

Figure 6-35. Format of the General LAN Information

Field	Type	Description
Length of general LAN information	BINARY(2)	The length of the general LAN information in the data unit. This must be set to 16.
Destination adapter address	CHAR(6)	Specifies, in packed form, the adapter address to which this data frame will be sent. **Note**: Because user-defined communications support only allows connectionless service over LANs, it is not necessary for all frames being sent on a single output operation to have the same destination adapter address.
DSAP address	CHAR(1)	The service access point on which the destination system will receive this frame. Any value may be used. **Note**: The Ethernet Version 2 standard does not use logical link control, which utilizes SAPs. Therefore, to send Ethernet Version 2 frames, a null DSAP address (X'00') must be specified in the DSAP address field. Also, the Ethernet Standard (ETHSTD) parameter in the Ethernet line description must be configured as either *ETHV2 or *ALL.
SSAP address	CHAR(1)	The service access point on which the AS/400 system will send this frame. Any service access point configured in the token ring or Ethernet line description may be used. **Note**: The Ethernet Version 2 standard does not use logical link control, which utilizes SAPs. Therefore, to send Ethernet Version 2 frames, a null SSAP address (X'00') must be specified in the SSAP address field. Also, the Ethernet Standard (ETHSTD) parameter in the Ethernet line description must be configured as either *ETHV2 or *ALL.
Access control	CHAR(1)	Specifies outbound frame priority and is mapped to the access priority bits in the access control field of 802.5 frames. For links using a token-ring communications line, any value between X'00' and X'07' may be used, where X'00' is the lowest priority and X'07' is the highest priority. For links using an Ethernet communications line, the content of this field is not applicable and must be set to X'00'.
Priority control	CHAR(1)	Specifies how to interpret the value set in the access control field. For links using a token-ring communications line, the valid values are as follows: **X'00'** Use any priority less than or equal to the value set in the access control field. **X'01'** Use the priority exactly equal to the value set in the access control field. **X'FF'** Use the AS/400 system default priority. For links using an Ethernet communications line, the content of this field is not applicable and must be set to X'00'.
Length of routing information	BINARY(2)	The length of the routing information in the data unit. For links using a token-ring communications line, any value between 0 and 18 may be used, where 0 indicates that there is no routing information. For links using an Ethernet communications line, the content of this field is not applicable and must be set to 0 indicating that there is no routing information.
Length of user data	BINARY(2)	The length of the user data in the data unit. This must be less than or equal to the maximum frame size allowed on the service access point specified in the SSAP address field. See "Query Line Description (QOLQLIND) API" on page 6-5 to determine the maximum frame size allowed on the service access point specified in the SSAP address field. For Ethernet Version 2 frames, this must be at least 48 and not more than 1502 (including 2 bytes for the Ethernet type field). **Note**: Ethernet 802.3 frames will be padded when the user data is less than 46 bytes.

Output Buffer Descriptor Element Format—LAN Operation X'0000': The information specified in each data unit of the output buffer must be described in the corresponding element of the output buffer descriptor. Figure 6-36 shows the format of each element in the output buffer descriptor.

Figure 6-36. Format of an Element in the Output Buffer Descriptor

Field	Type	Description
Length	BINARY(2)	The number of bytes of information in the corresponding data unit of the output buffer. This must be equal to the length of the general LAN information plus the length of the routing information plus the length of the user data. See Figure 6-35 on page 6-30 for more information on the format of the general LAN information.
Reserved	CHAR(30)	Reserved for extension.

X.25 SVC and PVC Output Operations

Figure 6-37 shows the output operations that are supported on links using an X.25 communications line.

Figure 6-37. X.25 SVC and PVC Output Operations

Operation	Meaning
X'0000'	Send user data (SVC or PVC). **Note**: This is a synchronous operation. Control will not return from the QOLSEND API until the operation completes.
X'B000'	Send a call request packet (SVC) or open the PVC connection. **Note**: This is an asynchronous operation. Notification of the completion of this operation will be returned from the QOLRECV API with operation X'B001' only after control returns from the QOLSEND API with a 0/0 return and reason code. See "Receive Data (QOLRECV) API" on page 6-14 for more information.
X'B100'	Send a clear packet (SVC) or close the PVC connection. **Note**: This is an asynchronous operation. Notification of the completion of this operation will be returned from the QOLRECV API with operation X'B101' only after control returns from the QOLSEND API with a 0/0 return and reason code. See "Receive Data (QOLRECV) API" on page 6-14 for more information.
X'B110'	Close all connections which were cleared by the reason given in the connection failure date received on X 'B311'. **Note**: This is an asynchronous operation. Notification of the completion of this operation will be returned from the QOLRECV API with operation X'B111' only after control returns from the QOLSEND API with a 0/0 return and reason code. See "Receive Data (QOLRECV) API" on page 6-14 for more information.
X'B400'	Send a call accept packet (SVC only). **Note**: This is a synchronous operation. Control will not return from the QOLSEND API until the operation completes.
X'BF00'	Send a reset request or reset confirmation packet (SVC or PVC). **Note**: This is an asynchronous operation. Notification of the completion of this operation will be returned from the QOLRECV API with operation X'BF01' only after control returns from the QOLSEND API with a 0/0 return and reason code. See "Receive Data (QOLRECV) API" on page 6-14 for more information.

Note: The maximum number of outstanding asynchronous operations (notification of completion not yet received from the QOLRECV API) is five. All calls made to the QOLSEND API or QOLSETF API under this condition will be rejected with a return and reason code of 83/3200.

X.25 Operation X'0000': This operation allows the application program to send user data on an SVC or PVC X.25 connection. The application must provide the following information:

- User data in the next data unit of the output buffer, starting with the first data unit
- A description, in the corresponding element of the output buffer descriptor, of the user data in that data unit.

For example, suppose a user-defined communications application program wants to send two data units of user data. The first portion of the user data would be placed in first data unit of the output buffer and described in the first element of the output buffer descriptor. The second portion of the user data would be placed in the second data unit of the output buffer and described in the second element of the output buffer descriptor. The number of data units parameter on the call to the QOLSEND API would be set to 2.

User-defined communications support automatically fragments the user data in each data unit into one or more appropriately sized X.25 packets based on the negotiated transmit packet size for the connection. All packets constructed for a data unit, except for the last (or only) packet, will always have the X.25 more data bit (M-bit) set on. See "Output Buffer Descriptor Element Format—X.25 Operation

X'0000'" on page 6-32 for more information on how to set the X.25 M-bit on or off in the last (or only) packet constructed for a data unit.

Data Unit Format—X.25 Operation X'000': Each data unit in the output buffer consists solely of user data and starts offset 0 from the top of the data unit.

Output Buffer Descriptor Element Format—X.25 Operation X'0000': The user data specified in each data unit of the output buffer must be described in the corresponding element of the output buffer descriptor.

Figure 6-38 shows the format of each element in the output buffer descriptor.

Figure 6-38. Format of an Element in the Output Buffer Descriptor

Field	Type	Description
Length	BINARY(2)	The number of bytes of user data in the corresponding data unit of the output buffer. This must always be less than or equal to the X.25 user data size parameter that was specified on the call to the QOLELINK API when the link was enabled. See "Enable Link (QOLELINK) API" on page 6-2 for more information.
More data indicator	CHAR(1)	Specifies whether the X.25 more data bit (M-bit) should be set on or off in the last (or only) X.25 packet constructed for the corresponding data unit. The valid values are as follows: **X'00'** Set the M-bit off in the last (or only) X.25 packet constructed for the corresponding data unit. **X'01'** Set the M-bit on in the last (or only) X.25 packet constructed for the corresponding data unit. **Note**: When this value is selected, the length field must be set to a multiple of the negotiated transmit packet size for the connection.
Qualified data indicator	CHAR(1)	Specifies whether the X.25 qualifier bit (Q-bit) should be set on or off in all X.25 packets constructed for the corresponding data unit. The valid values are as follows: **X'00'** Set the Q-bit off in all X.25 packets constructed for the corresponding data unit. **X'01'** Set the Q-bit on in all X.25 packets constructed for the corresponding data unit.
Interrupt packet indicator	CHAR(1)	Specifies whether the user data in the corresponding data unit should be sent in an X.25 interrupt packet. The valid values are as follows: **X'00'** Send the user data in the corresponding data unit in one or more X.25 data packets. **X'01'** Send the user data in the corresponding data unit in an X.25 interrupt packet. An interrupt packet causes the data to be expedited. **Note**: When this value is selected, the length field must be set to a value between 1 and 32, and the number of data units parameter on the call to the QOLSEND API must be set to 1. Also, the contents of the more data indicator, qualified data indicator, and delivery confirmation indicator fields are ignored.
Delivery confirmation indicator	CHAR(1)	Specifies whether the X.25 delivery confirmation bit (D-bit) should be set on or off in all X.25 packets constructed for the corresponding data unit. The valid values are as follows: **X'00'** Set the D-bit off in all X.25 packets constructed for the corresponding data unit. **X'01'** Set the D-bit on in all X.25 packets constructed for the corresponding data unit. **Note**: The AS/400 system does not fully support delivery confirmation when sending user data. Confirmation is from the local data circuit equipment (DCE).
Reserved	CHAR(26)	Reserved for extension.

X.25 Operation X'B000': This operation allows the application program to either initiate an SVC call or to open a PVC connection. The application must provide the data for this operation in the first data unit of the output buffer. The output buffer descriptor is not used.

The format of the data required for the X'B000' operation depends on whether it is used to initiate an SVC call or to open a PVC connection. Each format is explained in the following table.

Note: When initiating an SVC call, the AS/400 system chooses an available SVC to use. The logical channel identifier of the SVC that was chosen will be returned when notification of the completion of X'B000' is received from the QOLRECV API (operation X'B001'). See "Receive Data (QOLRECV) API" on page 6-14 for more information.

Data Unit Format—X.25 Operation X'B000' (Initiate an SVC Call): The data for this operation starts at offset 0 from the top of the first data unit in the output buffer.

Figure 6-39 shows the format of the data required for the X'B000' operation when initiating an SVC call.

Figure 6-39 (Page 1 of 3). Format of Data for X'B000' Operation (Initiate an SVC Call)

Field	Type	Description
Reserved	CHAR(1)	This field must be set to X'02'.

Figure 6-39 (Page 2 of 3). Format of Data for X'B000' Operation (Initiate an SVC Call)

Field	Type	Description
Reserved	CHAR(3)	This field must be set to hexadecimal zeros.
Transmit packet size	BINARY(2)	The requested transmit packet size for this connection. The valid values are 64, 128, 256, 512, 1024, 2048, and 4096. The value specified must be less than or equal to the transmit maximum packet size configured for this line. The special value of X'FFFF' may be specified to use the transmit default packet size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the transmit maximum packet size and the transmit default packet size configured for this line.
Transmit window size	BINARY(2)	The requested transmit window size for this connection. The valid values are as follows: **1–7** When modulus 8 is configured for this line. **1–15** When modulus 128 is configured for this line. **X'FFFF'** Use the transmit default window size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the modulus value and the transmit default window size configured for this line.
Receive packet size	BINARY(2)	The requested receive packet size for this connection. The valid values are 64, 128, 256, 512, 1024, 2048, and 4096. The value specified must be less than or equal to the receive maximum packet size configured for this line. The special value of X'FFFF' may be specified to use the receive default packet size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the receive maximum packet size and the receive default packet size configured for this line.
Receive window size	BINARY(2)	The requested receive window size for this connection. The valid values are as follows: **1–7** When modulus 8 is configured for this line. **1–15** When modulus 128 is configured for this line. **X'FFFF'** Use the receive default window size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the modulus value and the receive default window size configured for this line.
Reserved	CHAR(7)	This field must be set to hexadecimal zeros.
DTE address length	BINARY(1)	The number of binary coded decimal (BCD) digits in the DTE address to call. The valid values are as follows: **1–15** When extended network addressing is not configured for this line. **1–17** When extended network addressing is configured in the line description. See "Query Line Description (QOLQLIND) API" on page 6-5 to determine if extended network addressing is configured for this line.
DTE address	CHAR(16)	Specifies, in binary coded decimal (BCD), the DTE address to call. The address must be left justified and padded on the right with BCD zeros.
Reserved	CHAR(8)	This field must be set to hexadecimal zeros.
Delivery confirmation support	CHAR(1)	Specifies whether the X.25 delivery confirmation bit (D-bit) should be set on or off in the call request packet. The valid values are as follows: **X'00'** Set the D-bit off in the call request packet. **X'01'** Set the D-bit on in the call request packet.
Reserved	CHAR(7)	This field must be set to hexadecimal zeros.
Closed user group indicator	CHAR(1)	Specifies whether the closed user group (CUG) identifier should be included in the call packet. The valid values are as follows: **X'00'** Do not include the CUG identifier in the call packet. **X'01'** Include the CUG identifier in the call packet.
Closed user group identifier	CHAR(1)	The CUG identifier to be included in the call packet. The valid values are as follows: **X'00'** When the closed user group indicator field is set to X'00' **X'00'–X'99'** When the closed user group indicator field is set to X'01'
Reverse charging indicator	CHAR(1)	Specifies reverse charging options. The valid values are as follows: **X'00'** Do not request reverse charging. **X'01'** Request reverse charging.
Fast select indicator	CHAR(1)	Specifies fast select options. The valid values are as follows: **X'00'** Do not request fast select. **X'01'** Request fast select with restriction. **X'02'** Request fast select without restriction.

Figure 6-39 (Page 3 of 3). Format of Data for X'B000' Operation (Initiate an SVC Call)

Field	Type	Description
X.25 facilities length	BINARY(1)	The number of bytes of data in the X.25 facilities field. Any value between 0 and 109 may be used. **Note**: The AS/400 system codes the closed user group, reverse charging, and fast select facilities in the X.25 facilities field, if the user requested them in the above fields. Additionally, if the network user identification parameter (NETUSRID) is specified in the line description, the network user identification (NUI) facility is coded in the field, following the other additional facilities, if present. Finally, if the packet and window size values specified are different than the network default, the facilities containing these values are coded in the field as well. The system will update the X.25 facilities length field appropriately for each facility to which the AS/400 system adds the X.25 facilities field. This length cannot exceed 109 bytes.
X.25 facilities	CHAR(109)	Specifies additional X.25 facilities data requested. **Note**: The application programmer should not code the facilities for NUI, fast select, reverse charging, closed user group, packet size, or window size in this field. By doing so, this field could contain duplicate facilities, which may not be consistently supported by all X.25 networks.
Reserved	CHAR(48)	This field must be set to hexadecimal zeros.
Call user data length	BINARY(2)	The number of bytes of data in the call user data field. The valid values are as follows: **0—16** When the fast select indicator field is set to X'00'. **0—128** When the fast select indicator field is set to X'01' or X'02'.
Call user data	CHAR(128)	The call user data.
Reserved	CHAR(128)	This field must be set to hexadecimal zeros.
Control information	CHAR(1)	Specifies control information for this connection. This is a bit-sensitive field with bit 0 (leftmost bit) defined for reset support. The remaining bits are undefined and should be set off ('0'B). The valid values for bit 0 are as follows: **'0'B** Resets are not supported on this connection. When this value is selected, the X'BF00' output operation will not be valid on this connection. Also, a reset indication packet received on this connection will cause the connection to be ended. **'1'B** Resets are supported on this connection. When this value is selected, the X'BF00' output operation will be valid on this connection. Also, the user-defined communications application program will be required to handle reset indications received on this connection. For example, consider the following values for the control information field: **X'00'** Resets are not supported on this connection. **X'80'** Resets are supported on this connection.
Reserved	CHAR(3)	This field must be set to hexadecimal zeros.
Maximum data unit assembly size	BINARY(4)	The maximum number of bytes of user data that is received in a complete X.25 packet sequence before passing the user data to the application. Any value between 1024 and 32767 may be used, and should be set to the largest value that the application will support. **Notes**: 1. The system attempts to assemble the entire packet sequence before passing the data to the application. The only exception to this is when the size of the packet sequence exceeds the value the user specified for this field. 2. If the number of bytes of user data received in a complete X.25 packet sequence is more than can fit into one data unit of the input buffer, the more data indicator field in the corresponding element of the input buffer descriptor will be set to X'01' and the remaining user data will be filled in the next data unit. See "Receive Data (QOLRECV) API" on page 6-14 for more information. 3. There is no limitation on the number of bytes of user data that can be sent in a complete X.25 packet sequence. However, the QOLSEND API may need to called rnore than once.
Automatic flow control	BINARY(2)	Relates to the amount of data that will be held by user-defined communications support before sending a receive not ready (RNR) packet to the sending system. The recommended value for this field is 32, but any value between 1 and 128 may be used. **Note**: A receive ready (RR) packet will be sent when the user-defined communications application program receives some of the data.
Reserved	CHAR(30)	This field must be set to hexadecimal zeros.

Data Unit Format—X.25 Operation X'B000' (Open a PVC Connection): The data for this operation starts at offset 0 from the top of the first data unit in the output buffer.

Figure 6-40 shows the format of the data required for the X'B000' operation when opening a PVC connection

Figure 6-40 (Page 1 of 2). Format of Data for X'B000' Operation (Open a PVC Connection)

Field	Type	Description
Reserved	CHAR(1)	This field must be set to hexadecimal zeros.
Reserved	CHAR(1)	This field must be set to hexadecimal zeros.
Logical channel identifier	CHAR(2)	The logical channel identifier of the PVC to open. Any PVC configured for this line that is eligible to be used by the network controller that the link is using may be specified and must be in the range of X'0001'—X'0FFF'. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the PVCs configured for this line that are eligible to be used by the network controller the link is using.
Transmit packet size	BINARY(2)	The requested transmit packet size for this connection. The valid values are 64, 128, 256, 512, 1024, 2048, and 4096. The value specified must be less than or equal to the transmit maximum packet size configured for this line. The special value of X'FFFF' may be specified to use the transmit default packet size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the transmit maximum packet size and the transmit default packet size configured for this line.
Transmit window size	BINARY(2)	The requested transmit window size for this connection. The valid values are as follows: **1—7** When modulus 8 is configured for this line. **1—15** When modulus 128 is configured for this line. **X'FFFF'** Use the transmit default window size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the modulus value and the transmit default window size configured for this line.
Receive packet size	BINARY(2)	The requested receive packet size for this connection. The valid values are 64, 128, 256, 512, 1024, 2048, and 4096. The value specified must be less than or equal to the receive maximum packet size configured for this line. The special value of X'FFFF' may be specified to use the receive default packet size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the receive maximum packet size and the receive default packet size configured for this line.
Receive window size	BINARY(2)	The requested receive window size for this connection. The valid values are as follows: **1—7** When modulus 8 is configured for this line. **1—15** When modulus 128 is configured for this line. **X'FFFF'** Use the receive default window size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the modulus value and the receive default window size configured for this line.
Reserved	CHAR(32)	This field must be set to hexadecimal zeros.
Delivery confirmation support	CHAR(1)	The X.25 delivery confirmation bit (D-bit) support for this connection. The valid values are as follows: **X'00'** D-bit will be supported for sending data but not for receiving data. **Note**: When this value is selected and an X.25 packet is received with the D-bit set on, the input/output processor (IOP) will send a reset packet. **X'01'** D-bit will be supported for sending data and for receiving data.
Reserved	CHAR(427)	This field must be set to hexadecimal zeros.

Figure 6-40 (Page 2 of 2). Format of Data for X'B000' Operation (Open a PVC Connection)

Field	Type	Description
Control information	CHAR(1)	Specifies control information for this connection. This is a bit-sensitive field with bit 0 (leftmost bit) defined for reset support. The remaining bits are undefined and should be set off ('0'B). The valid values for bit 0 are as follows: **'0'B** Resets are not supported on this connection. When this value is selected, the X'BF00' output operation will not be valid on this connection. Also, a reset indication packet received on this connection will cause the connection to be ended. **'1'B** Resets are supported on this connection. When this value is selected, the X'BF00' output operation will be valid on this connection. Also, the user-defined communications application program will be required to handle reset indications received on this connection. For example, consider the following values for the control information field: **X'00'** Resets are not supported on this connection. **X'80'** Resets are supported on this connection.
Reserved	CHAR(3)	This field must be set to hexadecimal zeros.
Maximum data unit assembly size	BINARY(4)	The maximum number of bytes of user data that is received in a complete X.25 packet sequence before passing the user data to the application. Any value between 1024 and 32767 may be used, and should be set to the largest value that the application will support. **Notes**: 1. The system attempts to assemble the entire packet sequence before passing the data to the application. The only exception to this is when the size of the packet sequence exceeds the value the user specified for this field. 2. If the number of bytes of user data received in a complete X.25 packet sequence is more than can fit into one data unit of the input buffer, the more data indicator field in the corresponding element of the input buffer descriptor will be set to X'01' and the remaining user data will be filled in the next data unit. See "Receive Data (QOLRECV) API" on page 6-14 for more information. 3. There is no limit of the number of bytes of user data that can be sent in a complete X.25 packet sequence. However, the QOLSEND API may need to called more than once.
Automatic flow control	BINARY(2)	Relates to the amount of data that will be held by user-defined communications support before sending a receive not ready (RNR) packet to the sending system. The recommended value for this field is 32, but any value between 1 and 128 may be used. **Note**: A receive ready (RR) packet will be sent when the user-defined communications application program receives some of the data.
Reserved	CHAR(30)	This field must be set to hexadecimal zeros.

X.25 Operation X'B100': This operation allows the application program to either send a clear packet on an SVC, close an SVC connection that was cleared by the remote system, or to close a PVC connection. The application must provide the data for this operation in the first data unit of the output buffer. The output buffer descriptor is not used.

The format of the data required for the X'B100' operation is the same whether or not it is used to send a clear packet on an SVC or to close a PVC connection. The format of the data required for the X'B100' operation should be set to hexadecimal zeros if it is used to close an SVC connection that was previously cleared by the remote system.

Notes:

1. The AS/400 system provides the confirmation of the clear indication, however, the local user-defined communications application must issue the X'B100' operation to free the PCEP for the connection.
2. Closing a PVC connection will cause a reset packet to be sent to the remote system.

Data Unit Format—X.25 Operation X'B100': The data for this operation starts at offset 0 from the top of the first data unit in the output buffer.

Figure 6-41 shows the format of the data required for the X'B100' operation.

Figure 6-41 (Page 1 of 2). Format of Data for X'B100' Operation

Field	Type	Description
Reserved	CHAR(2)	This field must be set to hexadecimal zeros.

[1] This field is not used for PVC connections and should be set to hexadecimal zeros.

Figure 6-41 (Page 2 of 2). Format of Data for X'B100' Operation

Field	Type	Description
Cause code	CHAR(1)	The X.25 cause code.
Diagnostic code	CHAR(1)	The X.25 diagnostic code.
Reserved	CHAR(4)	This field must be set to hexadecimal zeros.
X.25 facilities length[1]	BINARY(1)	The number of bytes of data in the X.25 facilities field. Any value between 0 and 109 may be used.
X.25 facilities[1]	CHAR(109)	The X.25 facilities data.
Reserved	CHAR(48)	This field must be set to hexadecimal zeros.
Clear user data length[1]	BINARY(2)	The number of bytes of data in the clear user data field. Any value between 0 and 128 may be used.
Clear user data[1]	CHAR(128)	The clear user data. **Note**: The CCITT standard recommends that this field only be present in conjunction with the fast select or call deflection selection facility. The AS/400 system does not enforce this restriction, however.
Reserved	CHAR(216)	This field must be set to hexadecimal zeros.

[1] This field is not used for PVC connections and should be set to hexadecimal zeros.

X.25 Operation X'B110': This operation allows the application program to clean up all internal control information on all the connections over the link and free up all PCEPs and UCEPs. This operation is only valid following the receipt of the X'B311' operation that reports the connection failure data to the application. There is no data associated with this operation.

X.25 Operation X'B400': This operation allows the application program to accept an incoming SVC call. The application must provide the data for this operation in the first data unit of the output buffer. The output buffer descriptor is not used.

Note: Notification of incoming calls are received from the QOLRECV API with operation X'B201'. See "Receive Data (QOLRECV) API" on page 6-14 for more information.

Data Unit Format—X.25 Operation X'B400': The data for this operation starts at offset 0 from the top of the first data unit in the output buffer.

Figure 6-42 shows the format of the data required for the X'B400' operation.

Figure 6-42 (Page 1 of 3). Format of Data for X'B400' Operation

Field	Type	Description
Reserved	CHAR(1)	This field must be set to hexadecimal zeros.
Reserved	CHAR(3)	This field must be set to hexadecimal zeros.
Transmit packet size	BINARY(2)	The transmit packet size for this connection. The valid values are 64, 128, 256, 512, 1024, 2048, and 4096. The value specified must be less than or equal to the transmit maximum packet size configured for this line. The special value of X'FFFF' may be specified to use the transmit default packet size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the transmit maximum packet size and the transmit default packet size configured for this line.
Transmit window size	BINARY(2)	The transmit window size for this connection. The valid values are as follows: **1—7** When modulus 8 is configured for this line. **1—15** When modulus 128 is configured for this line. **X'FFFF'** Use the transmit default window size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the modulus value and the transmit default window size configured for this line.

Figure 6-42 (Page 2 of 3). Format of Data for X'B400' Operation

Field	Type	Description
Receive packet size	BINARY(2)	The receive packet size for this connection. The valid values are 64, 128, 256, 512, 1024, 2048, and 4096. The value specified must be less than or equal to the receive maximum packet size configured for this line. The special value of X'FFFF' may be specified to use the receive default packet size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the receive maximum packet size and the receive default packet size configured for this line.
Receive window size	BINARY(2)	The receive window size for this connection. The valid values are as follows: **1—7** When modulus 8 is configured for this line. **1—15** When modulus 128 is configured for this line. **X'FFFF'** Use the receive default window size configured for this line. See "Query Line Description (QOLQLIND) API" on page 6-5 for information on determining the modulus value and the receive default window size configured for this line.
Reserved	CHAR(32)	This field must be set to hexadecimal zeros.
Delivery confirmation support	CHAR(1)	Specifies whether the X.25 delivery confirmation bit (D-bit) should be set on or off in the call accept packet. This also specifies the D-bit support for this connection. The valid values are as follows: **X'00'** Set the D-bit off in the call accept packet. D-bit will be supported for sending data but not for receiving data. **Note**: When this value is selected and an X.25 packet is received with the D-bit set on, the input/output processor (IOP) will send a reset packet. **X'01'** Set the D-bit on in the call accept packet. D-bit will be supported for sending data and for receiving data.
Reserved	CHAR(11)	This field must be set to hexadecimal zeros.
X.25 facilities length	BINARY(1)	The number of bytes of data in the X.25 facilities field. Any value between 0 and 109 may be used. **Note**: The AS/400 system codes the packet and window size facilities in this field, if necessary. The total length of all facilities can not exceed 109 bytes.
X.25 facilities	CHAR(109)	The X.25 facilities data. **Note**: The application programmer should not code the facilities for packet or window sizes in this field. By doing so, this field could contain duplicate facilities, which may not be consistently supported by all X.25 networks.
Reserved	CHAR(306)	This field must be set to hexadecimal zeros.
Control information	CHAR(1)	Specifies control information for this connection. This is a bit-sensitive field with bit 0 (leftmost bit) defined for reset support. The remaining bits are undefined and should be set off ('0'B). The valid values for bit 0 are as follows: **'0'B** Resets are not supported on this connection. When this value is selected, the X'BF00' output operation will not be valid on this connection. Also, a reset indication packet received on this connection will cause the connection to be ended. **'1'B** Resets are supported on this connection. When this value is selected, the X'BF00' output operation will be valid on this connection. Also, the user-defined communications application program will be required to handle reset indications received on this connection. For example, consider the following values for the control information field: **X'00'** Resets are not supported on this connection. **X'80'** Resets are supported on this connection.
Reserved	CHAR(3)	This field must be set to hexadecimal zeros.

Figure 6-42 (Page 3 of 3). Format of Data for X'B400' Operation

Field	Type	Description
Maximum data unit assembly size	BINARY(4)	The maximum number of bytes of user data that can be received in a complete X.25 packet sequence on this connection. If this limit is exceeded, the connection will be ended. Any value between 1024 and 32767 may be used. **Notes**: 1. If the number of bytes of user data received in a complete X.25 packet sequence is more than can fit into one data unit of the input buffer, the more data indicator field in the corresponding element of the input buffer descriptor will be set to X'01' and the remaining user data will be filled in the next data unit. See "Receive Data (QOLRECV) API" on page 6-14 for more information. 2. There is no limitation on the number of bytes of user data that can be sent in a complete X.25 packet sequence. However, the QOLSEND API may need to called more than once.
Automatic flow control	BINARY(2)	Relates to the amount of data that will be held by user-defined communications support before sending a receive not ready (RNR) packet to the sending system. The recommended value for this field is 32, but any value between 1 and 128 may be used. **Note**: A receive ready (RR) packet will be sent when the user-defined communications application program receives some of the data.
Reserved	CHAR(30)	This field must be set to hexadecimal zeros.

X.25 Operation X'BF00': This operation allows an application program to send a reset request packet or a reset confirmation packet on an X.25 SVC or PVC connection. The application must provide the X.25 cause and diagnostic codes required for this operation in the first data unit of the output buffer. The output buffer descriptor is not used.

Information indicating whether a reset request or reset confirmation packet was sent is returned when notification of the completion of the X'BF00' operation is received from the QOLRECV API (operation X'BF01'). This information will be in the diagnostic data parameter of the QOLRECV API. See "Receive Data (QOLRECV) API" on page 6-14 for more information.

A reset confirmation packet will be sent under the following conditions:

- After a reset indication packet has been received on the connection and the application has received it from the QOLRECV API (X'B301' operation, 83/3202 return and reason code)
- After a reset indication packet has been received on the connection but before the application has received it from the QOLRECV API
- When a reset indication packet is received on the connection at the same time the X'BF00' output operation is issued

  This is known as a reset collision. In this case, user-defined communications support will discard the reset indication and, therefore, the application program will not receive it from the QOLRECV API. However, the cause and diagnostic codes from the reset indication are returned in the diagnostic data parameter of the QOLRECV program when the application receives notification of the completion of the X'BF00' operation. See "Receive Data (QOLRECV) API" on page 6-14 for more information.

A reset request packet will be sent when none of the above conditions are true.

Notes:

1. Data not yet received by the application program on a connection will *not* be deleted when a X'BF00' operation is issued on that connection. This data will be received before the notification of the completion of the X'BF00' operation is received from the QOLRECV API (operation X'BF01'). Data received after the notification of the completion of the X'BF00' operation is received should be treated as new data.
2. The X'BF00' operation is only valid on connections that support resets. See "X.25 Operation X'B000'" on page 6-32 and "X.25 Operation X'B400'" on page 6-37 for more information on specifying reset support.

Data Unit Format—X.25 Operation X'BF00': The first 2 bytes of the data unit in the output buffer are used for this operation. The first byte contains the X.25 cause code. The second byte contains the X.25 diagnostic code.

Return and Reason Codes

The return and reason codes that can be returned from the QOLSEND API depend on the type of communications line the link is using and on the operation that was requested.

LAN Return and Reason Codes

Return and Reason Codes for LAN Operation X'0000'

Figure 6-43. Return and Reason Codes for LAN Operation X'0000'

Return / Reason Code	Meaning	Recovery
0/0	Operation successful.	Continue processing.
80/2200	Queue error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Then correct the error, enable the link, and try the request again.
80/2401	Output buffer or output buffer descriptor error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Then correct the error, enable the link, and try the request again.
80/3002	A previous error occurred on this link that was reported to the application program by escape message CPF91F0 or CPF91F1. However, the application program has attempted another operation.	Ensure the link is disabled and see messages in the job log for further information. If escape message CPF91F0 was sent to the application program, then report the problem using the ANZPRB command. Otherwise, correct the error, enable the link, and try the request again.
80/4000	Error recovery has been canceled for this link.	Ensure the link is disabled and see messages in the job log for further information. Correct the condition, enable the link, and try the request again.
80/8000	The amount of user data in a data unit of the output buffer is greater than the maximum frame size allowed on the communications line the link is using. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled. Correct the error, enable the link, and try the request again.
80/9999	Internal system error detected. Escape message CPF91F0 will be sent to the application program when this return and reason code is received.	See messages in the job log for further information. Report the problem using the ANZPRB command.
83/1006	Output operation not valid.	Correct the operation parameter. Try the request again.
83/1007	Connection identifier not valid.	Correct the existing provider connection end point ID parameter. Try the request again.
83/1008	Number of data units not valid.	Correct the number of data units parameter. Try the request again.
83/1998	The amount of data in a data unit of the output buffer is not correct.	Correct the amount of user data, or the total amount of general LAN information, routing information, and user data in the offending data unit. Try the request again.
83/1999	Incorrect data in a data unit of the output buffer. The error offset field in the diagnostic data parameter will point to the incorrect data.	Correct the incorrect data. Try the request again.
83/3001	Link not enabled.	Correct the communications handle parameter. Try the request again.
83/3004	Link is enabling.	Wait for the enable-complete entry to be sent to the data queue or user queue. If the link was successfully enabled, try the request again.
83/4001	Link failure, system starting error recovery for this link.	Wait for the link to recover. Try the request again.
83/4003	Error detected by the input/output processor (IOP). The diagnostic data parameter will contain more information on this error.	Correct the error, and try the request again.

X.25 Return and Reason Codes

General X.25 Return and Reason Codes: Figure 6-44 shows the return and reason codes that can be received from the QOLSEND API for any requested operation.

Figure 6-44. Return and Reason Codes Valid for All X.25 Operations

Return / Reason Code	Meaning	Recovery
80/2200	Queue error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Correct the error, enable the link, and try the request again.
80/2401	Output buffer or output buffer descriptor error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Correct the error, enable the link, and try the request again.
80/3002	A previous error occurred on this link that was reported to the application program by escape message CPF91F0 or CPF91F1. However, the application has attempted another operation.	Ensure the link is disabled and see messages in the job log for further information. If escape message CPF91F0 was sent to the application program, report the problem using the ANZPRB command. Otherwise, correct the error, enable the link, and try the request again.
80/4000	Error recovery has been canceled for this link.	Ensure the link is disabled and see messages in the job log for further information. Correct the condition, enable the link, and try the request again.
80/9999	Internal system error detected. Escape message CPF91F0 will be sent to the application program when this return and reason code is received.	See messages in the job log for further information. Report the problem using the ANZPRB command.
83/1006	Output operation not valid.	Correct the operation parameter. Try the request again.
83/3001	Link not enabled.	Correct the communications handle parameter. Try the request again.
83/3004	Link is enabling.	Wait for the enable-complete entry to be sent to the data queue or user queue. If the link was successfully enabled, try the request again.
83/3200	All resources are currently in use by asynchronous operations that have not yet completed.	Wait for at least one of the asynchronous operations to complete. Notification of completion of these operations will be received from the QOLRECV API. Try the request again.

Return and Reason Codes for X.25 Operation X'0000'

Figure 6-45 (Page 1 of 2). Return and Reason Codes for X.25 Operation X'0000'

Return / Reason Code	Meaning	Recovery
0/0	Operation successful.	Continue processing.
83/1007	Connection identifier not valid.	Correct the existing provider connection end point ID parameter. Try the request again.
83/1008	Number of data units not valid.	Correct the number of data units parameter. Try the request again.
83/1997	The amount of user data in a data unit of the output buffer is not a multiple of the negotiated transmit packet size, and the more data indicator in the corresponding element of the output buffer descriptor is set to X'01'.	Correct the amount of user data in the offending data unit. Try the request again.
83/1998	The amount of user data in a data unit of the output buffer is not correct.	Correct the amount of user data in the offending data unit. Try the request again.
83/3201	The maximum amount of incoming user data that can be held by user-defined communications support for the application program on this connection has been exceeded.	Wait to receive a failure notification from the QOLRECV API indicating this condition (X'B301' operation, 83/3201 return and reason code). Issue the X'B100' output operation to end the connection.
83/3202	A reset indication has been received on this connection. The X.25 cause and diagnostic code fields in the diagnostic data parameter will contain the cause and diagnostic codes of the reset indication.	Wait to receive notification from the QOLRECV API indicating this condition (X'B301' operation, 83/3202 return and reason code). Issue the X'BF00' output operation to send a reset confirmation packet.
83/3205	Connection not in a valid state.	Ensure the connection is in a valid state for this operation. Try the request again.

Figure 6-45 (Page 2 of 2). Return and Reason Codes for X.25 Operation X'0000'

Return / Reason Code	Meaning	Recovery
83/4001	Link failure, system starting error recovery for this link.	Wait to receive a failure notification from the QOLRECV API indicating this condition (X'B301' or X'B311' operation, 83/4001 return and reason code). Issue the X'B100' output operation to end the connection.
83/4002	Connection failure.	Wait to receive a failure notification from the QOLRECV API indicating this condition (X'B301' operation, 83/4002 return and reason code). Issue the X'B100' output operation to end the connection.
83/4003	Data not sent. Error detected by input/output processor.	Try the request again. If the error persists, use the ANZPRB command to analyze and report the problem.

Return and Reason Codes for X.25 Operation X'B000'

Figure 6-46. Return and Reason Codes for X.25 Operation X'B000'

Return / Reason Code	Meaning	Recovery
0/0	Operation initiated.	Wait for notification of the completion of the X'B000' operation from the QOLRECV API (X'B001' operation).
83/4005	All connections are currently in use.	Wait for a connection to become available and try the request again.

Return and Reason Codes for X.25 Operation X'B100'

Figure 6-47. Return and Reason Codes for X.25 Operation X'B100'

Return / Reason Code	Meaning	Recovery
0/0	Operation initiated.	Wait for notification of the completion of the X'B100' operation from the QOLRECV API (X'B101' operation).
83/1007	Connection identifier not valid.	Correct the existing provider connection end point ID parameter. Try the request again.
83/3205	Connection not in a valid state.	Ensure the connection is in a valid state for this operation. Try the request again.

Return and Reason Codes for X.25 Operation X'B110'

Figure 6-48. Return and Reason Codes for X.25 Operation X'B110'

Return / Reason Code	Meaning	Recovery
0/0	Operation initiated.	Wait for notification of the completion of the X'B110' operation from the QOLRECV API (X'B111' operation).

Return and Reason Codes for X.25 Operation X'B400'

Figure 6-49 (Page 1 of 2). Return and Reason Codes for X.25 Operation X'B400'

Return / Reason Code	Meaning	Recovery
0/0	Operation successful.	Continue processing.
83/1007	Connection identifier not valid.	Correct the existing provider connection end point ID parameter. Try the request again.

Figure 6-49 (Page 2 of 2). Return and Reason Codes for X.25 Operation X'B400'

Return / Reason Code	Meaning	Recovery
83/1999	Incorrect data in a data unit of the output buffer. The error offset field in the diagnostic data parameter will point to the incorrect data.	Correct the incorrect data. Try the request again.
83/3205	Connection not in a valid state.	Ensure the connection is in a valid state for this operation. Try the request again.
83/4001	Link failure, system starting error recovery for this link.	Issue the X'B100' output operation to end the connection.
83/4004	Inbound call timed out.	Issue the X'B100' output operation to end the connection.

Return and Reason Codes for X.25 Operation X'BF00'

Figure 6-50. Return and Reason Codes for X.25 Operation X'BF00'

Return / Reason Code	Meaning	Recovery
0/0	Operation initiated.	Wait for notification of the completion of the X'BF00' operation from the QOLRECV API (X'BF01' operation).
83/1007	Connection identifier not valid.	Correct the existing provider connection end point ID parameter. Try the request again.
83/3205	Connection not in a valid state.	Ensure the connection is in a valid state for this operation. Try the request again.

Error Messages

CPF91F0 E Internal system error.
CPF91F1 E User-defined communications application error.

Set Filter (QOLSETF) API

Parameters

Required Parameter Group:

1	Return code	Output	Binary(4)
2	Reason code	Output	Binary(4)
3	Error offset	Output	Binary(4)
4	Communications handle	Input	Char(10)

The Set Filter (QOLSETF) API activates and/or deactivates one or more filters for a link that is currently enabled in the job in which the application program is running. The application program must provide the required filter information in the output buffer that was created when the link was enabled. The output buffer descriptor is not used. See "Format of Filter Information" on page 6-44 for details on the format of the filter information in the output buffer.

Filters contain inbound routing information that user-defined communications support uses to route incoming data to a link that is enabled by an application program. The incoming data that is routed depends on the type of communications line the link is using. On an X.25 communications line, the incoming data is an incoming switched virtual circuit (SVC) call. On a token-ring or Ethernet communications line, the incoming data is the actual data frame.

The type of filters activated for a link determine the way incoming data is routed to that link[1]. For links using a token-ring or Ethernet communications line, there are three types of filters. The following list of filters is from most to least restrictive:

- Destination service access point (DSAP), source service access point (SSAP), optional frame type, and sending adapter address
- DSAP, SSAP, and optional frame type
- DSAP

For links using an X.25 communications line, there are two types of filters. The following list of filters is from most to least restrictive:

- Protocol identifier (PID) and calling data terminal equipment (DTE) address

  The AS/400 system treats the first byte of call-user data in an X.25 call request packet as the PID.
- PID

The order for checking filters when multiple links are using the same communications line, is from most to least restrictive. For example, suppose two user-defined communications application programs (application program A

[1] All active filters for a link must be of the same type.

and B) in different jobs each have a link enabled that use the same token-ring communications line. Further suppose that application program A has activated a filter on DSAP X'22' and application program B has activated a filter on DSAP X'22' and SSAP X'22'. If a data frame comes in with a DSAP of X'22' and an SSAP of X'22', application program B will receive the frame. If a data frame comes in with a DSAP of X'22' and an SSAP not equal to X'22', application program A will receive the frame.

Required Parameter Group

Return code
OUTPUT: BINARY(4)
The recovery action to take. See "Return and Reason Codes" on page 6-46.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-46.

Error offset
OUTPUT; BINARY(4)
The offset from the top of the output buffer to the incorrect filter header data or to the incorrect filter in the filter list.
The content of this parameter is only valid for 83/1999 and 83/3003 return/reason codes.

Communications handle
INPUT: CHAR(10)
The name of the link on which to perform the filter operation.

Format of Filter Information

The application must provide all filter information in the output buffer that was created when the link was enabled. The application should treat the output buffer as one large space with the size equal to the number of data units created for the output buffer multiplied by the size of each data unit. This information is returned by the QOLELINK API when the link was enabled.

The filter information in the output buffer is made up of two parts. The first portion starts at offset 0 from the top of the output buffer and contains filter header data. The second portion of the filter information starts immediately after the filter header data in the output buffer and contains the filters that make up the filter list.

Figure 6-51. Filter Header Data

Field	Type	Description
Function	CHAR(1)	The filter function to perform. The valid values are as follows: **X'00'** Deactivate all filters that are currently active for this link and activate the filters specified in the filter list for this link. **X'01'** Activate the filters specified in the filter list for this link. All filters currently active for this link will remain active. **X'02'** Deactivate the filters specified in the filter list that are currently active for this link.
Filter type	CHAR(1)	The type of the filters in the filter list. All filters in the filter list must be of this type. In addition, this must be the same type as the filters currently active for this link, if any. The valid values are as follows: **X'00'** PID. This filter type is only applicable for links using an X.25 communications line and only applies to incoming SVC calls. **X'01'** PID and calling DTE address. This filter type is only applicable for links using an X.25 communications line and only applies to incoming SVC calls. **X'02'** DSAP. This filter type is only applicable for links using a token-ring or Ethernet communications line. **X'03'** DSAP, SSAP, and optional frame type. This filter type is only applicable for links using a token-ring or Ethernet communications line. **X'04'** DSAP, SSAP, optional frame type, and sending adapter address. This filter type is only applicable for links using a token-ring or Ethernet communications line. **Note**: The filter type field must be set even if there are no filters in the filter list.
Number of filters	BINARY(2)	The number of filters in the filter list. Any value between 0 and 256 may be used. **Note**: The maximum number of filters that can be specified in the filter list is also limited by the total size of the output buffer which may accommodate less than 256 filters.
Filter length	BINARY(2)	The length of each filter in the filter list. This value must be 16 for filter types X'00' and X'01', and 14 for filter types X'02', X'03', and X'04'. **Note**: The filter length field must be set even if there are no filters in the filter list.

The format of each filter in the previous list of filters is described in the following table. All filters in the list of filters must be contiguous with each other and be of the

type specified in the filter type field in the filter header data.

X.25 Filters (Filter Types X'00' and X'01')

Figure 6-52. Filter Types X'00' and X'01'

Field	Type	Description
PID length	CHAR(1)	The length of the PID on which to route incoming calls. The valid values are as follows: **X'00'** Route incoming calls with no PID specified. That is, with no call user data in the call request packet. **X'01'** Route incoming calls with the PID being treated as the first byte of call user data in the call request packet.
PID	CHAR(1)	The PID on which to route incoming calls. This should be set to X'00' when the PID length field is set to X'00'. Otherwise, any value may be used. **Note**: Care should be taken when setting the PID field to an SNA PID (X'C3', X'C6', X'CB', X'CE'), asynchronous PID (X'01', X'C0'), or TCP/IP PID (X'CC'). See the *X.25 Network Guide* for more information.
Calling DTE address length	CHAR(1)	Specifies, in hexadecimal, the number of binary coded decimal (BCD) digits in the calling DTE address on which to route incoming calls. The valid values are as follows: **X'00'** For filter type X'00'. **X'01'–X'0F'** For filter type X'01' when extended network addressing is not configured in the line description. See "Query Line Description (QOLQLIND) API" on page 6-5 to determine if extended network addressing is configured for this line. **X'01'–X'11'** For filter type X'01' when extended network addressing is configured in the line description. See "Query Line Description (QOLQLIND) API" on page 6-5 to determine if extended network addressing is configured for this line.
Calling DTE address	CHAR(12)	Specifies, in binary coded decimal (BCD), the calling DTE address on which to route incoming calls. This should be set to BCD zeros when the calling DTE address length field is set to X'00'. Otherwise, any valid DTE address left-justified and padded on the right with BCD zeros may be used.
Additional routing data	CHAR(1)	Specifies additional data on which to route incoming calls. This field is applicable for all X.25 filter types and is bit-sensitive with bit 0 (leftmost bit) defined for reverse charging options and bit 1 defined for fast select options. The remaining bits are undefined and should be set off ('0'B). The valid values for bit 0 are as follows: **'0'B** Accept reverse charging. **'1'B** Do not accept reverse charging. The valid values for bit 1 are as follows: **'0'B** Accept fast select. **'1'B** Do not accept fast select. For example, consider the following values for the additional routing data field: **X'00'** Accept reverse charging and accept fast select. **X'40'** Accept reverse charging and do not accept fast select. **X'80'** Do not accept reverse charging and accept fast select. **X'C0'** Do not accept reverse charging and do not accept fast select.

Token-ring and Ethernet Filters (Filter Types X'02', X'03', and X'04')

Figure 6-53 (Page 1 of 2). Filter Types X'02', X'03', and X'04'

Field	Type	Description
DSAP address length	CHAR(1)	The length of the DSAP address on which to route incoming frames. This must be set to X'01'.
DSAP address	CHAR(1)	The DSAP address on which to route incoming frames. The DSAP address is the service access point on which the incoming frame arrived. Any service access point configured in the token-ring or Ethernet line description as *NONSNA may be used. **Note**: The Ethernet Version 2 standard does not use logical link control, which utilizes SAPs. Therefore, to receive Ethernet Version 2 frames, a null DSAP address (X'00') must be specified in the DSAP address field. Also, the Ethernet Standard (ETHSTD) parameter in the Ethernet line description must be configured as either *ETHV2 or *ALL.

Figure 6-53 (Page 2 of 2). Filter Types X'02', X'03', and X'04'

Field	Type	Description
SSAP address length	CHAR(1)	The length of the SSAP address on which to route incoming frames. The valid values are as follows: **X'00'** For filter type X'02'. **X'01'** For filter types X'03' and X'04'.
SSAP address	CHAR(1)	The SSAP address on which to route incoming frames. The SSAP address is the service access point on which the incoming frame was sent. The valid values are as follows: **X'00'** For filter type X'02'. **X'00'−X'FF'.** For filter types X'03' and X'04'. **Note**: The Ethernet Version 2 standard does not use logical link control, which utilizes SAPs. Therefore, to receive Ethernet Version 2 frames, a null SSAP address (X'00') must be specified in the SSAP address field. Also, the Ethernet Standard (ETHSTD) parameter in the Ethernet line description must be configured as either *ETHV2 or *ALL.
Frame type length	CHAR(1)	The length of the frame type on which to route incoming frames. The valid values are as follows: **X'00'** For filter type X'02'. Also for filter types X'03' and X'04' when the DSAP address and SSAP address fields are not both set to X'00'. **X'00' or X'02'** For filter types X'03' and X'04' when the 'DSAP address' and SSAP address fields are both set to X'00'.
Frame type	CHAR(2)	The frame type on which to route incoming frames. The frame type is defined in an Ethernet Version 2 frame to indicate the upper layer protocol being used. This must be set to X'0000' when the frame type length field is set to X'00'. Otherwise, any value except X'80D5' (encapsulated LLC) may be used, but should be in the range of X'05DD'−X'FFFF'.
Sending adapter address length	CHAR(1)	Specifies, in hexadecimal, the length of the sending adapter address on which to route incoming frames. The valid values are as follows: **X'00'** For filter types X'02' and X'03'. **X'06'** For filter type X'04'.
Sending adapter address	CHAR(6)	Specifies, in packed form, the sending adapter address on which to route incoming frames. This must be set to X'000000000000' when the sending adapter address length field is set to X'00'. Otherwise, any valid adapter address may be used.

General Rules for Using Filters

The following is a list of rules for activating and deactivating filters:

- All active filters for a link must be of the same type
- A link can have a maximum of 256 active filters
- The maximum number of filters that can be specified in the filter list can be no more than 256, and may be less, depending on the size of the output buffer
- A request to activate a filter for a link that already has the same filter active will be successful, but the filter will only be activated once
- A request to deactivate a filter for a link that has no such filter active will be successful
- If the return and reason code from the QOLSETF API is not 0/0, none of the specified filters were activated or deactivated
- Once a filter is activated, it will remain active until one of the following occurs:
  - It is deactivated by explicitly calling the QOLSETF API
  - The link that the filter was active for is disabled

Return and Reason Codes

Figure 6-54 (Page 1 of 2). Return and Reason Codes for the QOLSETF API

Return / Reason Code	Meaning	Recovery
0/0	Operation successful.	Continue processing.
80/2200	Queue error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Then correct the error, enable the link, and try the request again.
80/2401	Output buffer error detected. Escape message CPF91F1 will be sent to the application program when this return and reason code is received.	Ensure the link is disabled and see messages in the job log for further information. Then correct the error, enable the link, and try the request again.

Figure 6-54 (Page 2 of 2). Return and Reason Codes for the QOLSETF API

Return / Reason Code	Meaning	Recovery
80/3002	A previous error occurred on this link that was reported to the application program by escape message CPF91F0 or CPF91F1. However, the application program has attempted another operation.	Ensure the link is disabled and see messages in the job log for further information. If escape message CPF91F0 was sent to the application program, then report the problem using the ANZPRB command. Otherwise, correct the error, enable the link, and try the request again.
80/4000	Error recovery has been canceled for this link.	Ensure the link is disabled and see messages in the job log for further information. Then correct the condition, enable the link, and try the request again.
80/9999	Internal system error detected. Escape message CPF91F0 will be sent to the application program when this return and reason code is received.	See messages in the job log for further information. Then, report the problem using the ANZPRB command.
83/1998	The size of the output buffer is not large enough for the specified number of filters.	Reduce the number of filters in the filter list so that the size of the filter list plus the size of the filter header data is less than or equal to the size of the output buffer. Try the request again.
83/1999	Incorrect filter header data or incorrect filter in the filter list. If the filter header data is incorrect, the error offset parameter will point to the field in error. If a filter in the filter list is incorrect, the error offset parameter will point to the beginning of the incorrect filter.	Correct the incorrect filter header data or the incorrect filter in the filter list. Try the request again.
83/3001	Link not enabled.	Correct the communications handle parameter. Try the request again.
83/3003	One of the following is true of a filter in the filter list. The error offset parameter will point to the beginning of the offending filter. • The filter is already activated by another job using the same communications line • The service access point, specified in the DSAP address field of the filter, is not configured in the token-ring or Ethernet line description • The DSAP address field of the filter contains the null DSAP address (X'00'), but the Ethernet Standard (ETHSTD) parameter in the Ethernet line description is not configured as *ETHV2 or *ALL • The service access point, specified in the DSAP address field of the filter, is configured in the token-ring or Ethernet line description for SNA use only (*SNA)	Do one of the following, and try the request again: • End the job that has already activated the filter • Configure the service access point in the token-ring or Ethernet line description • Delete the Ethernet line description, and create another Ethernet line description specifying *ETHV2 or *ALL in the Ethernet Standard (ETHSTD) parameter • Change the service access point in the token-ring or Ethernet line description to non-SNA use (*NONSNA)
83/3004	Link is enabling.	Wait for the enable-complete entry to be sent to the data queue or user queue. If the link was successfully enabled, try the request again.
83/3200	All resources are currently in use by asynchronous operations that have not yet completed. **Note**: This return and reason code is only possible for links using an X.25 communications line. See "Send Data (QOLSEND) API" on page 6-27 for more information.	Wait for at least one of the asynchronous operations to complete. Notification of completion of these operations will be received from the QOLRECV API. Try the request again.
83/4001	Link failure, system starting error recovery for this link.	Wait for the link to recover. Try the request again.

Error Messages

CPF91F0 E Internal system error.

CPF91F1 E User-defined communications application error.

Set Timer (QOLTIMER) API

Parameters

Required Parameter Group:

1	Return code	Output	Binary(4)
2	Reason code	Output	Binary(4)
3	Timer set	Output	Char(8)
4	Timer to cancel	Input	Char(8)
5	Queue name	Input	Char(20)
6	Operation	Input	Char(1)
7	Interval	Input	Binary(4)
8	Establish count	Input	Binary(4)
9	Key length	Input	Binary(4)
10	Key value	Input	Char(256)
11	User data	Input	Char(60)

Optional Parameter:

12	Queue type	Input	Char(1)

The Set Timer (QOLTIMER) API either sets or cancels a timer. Up to 128 timers, each uniquely identified by a name (timer handle), can be set in the job in which the application program is running.

When the QOLTIMER API is called to set a timer, a timer handle is returned to the application program. The timer handle, along with the user data supplied when the timer was set, is included in the timer-expired entry that is sent to the data queue or user queue when the specified amount of time for this timer has elapsed. The timer is then reestablished, if necessary.

For example, suppose a user-defined communications application program sets a timer with a five-second interval to be established two times. After five seconds, the timer-expired entry for this timer will be sent to the data queue or user queue specified when the timer was set. The timer will then be automatically reestablished, and five seconds later, another timer-expired entry for this timer will be sent to the data queue or user queue. See "Timer-Expired Entry" on page 5-3 for the format of the timer-expired entry.

In addition to setting a timer, the application program can call the QOLTIMER API to cancel one or all timers currently set in the job in which the application program is running. User-defined communications support will implicitly cancel a timer in the following cases:

- After a timer has expired the specified number of times
- When a job ends that had one or more timers set

Note: User-defined communications support does not associate timers with links. If necessary, that association must be done by the application.

Required Parameter Group

Return code
OUTPUT; BINARY(4)
The recovery action to take. See "Return and Reason Codes" on page 6-49.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-49.

Timer set
OUTPUT; CHAR(8)
The name of the timer (timer handle) that was set. TIMER001, TIMER002, ... , TIMER128 are the possible values.

The content of this parameter is only valid when setting a timer.

Timer to cancel
INPUT; CHAR(8)
The name of the timer (timer handle) to cancel. TIMER001, TIMER002, ... , TIMER128 may be used as values. The special value of *ALL (left-justified and padded on right with spaces) may be used to cancel all timers currently set in the job in which the user-defined communications application program is running.

The content of this parameter is only valid when canceling a timer.

Queue name
INPUT; CHAR(20)
The name and library of the data queue or user queue where the timer-expired entry will be sent when the timer expires. The first 10 characters specify the name of the data queue or user queue and the second 10 characters specify the library in which the queue is located. Both entries are left-justified. The special values of *LIBL and *CURLIB may be used for the library name.

The content of this parameter is only valid when setting a timer.

Operation
INPUT; CHAR(1)
The timer operation to perform. The valid values are as follows:

X'01' Set a timer.
X'02' Cancel a timer.

Interval
INPUT; BINARY(4)
The number of milliseconds for which to set this timer. Any value between 1,048 and 3,600,000 may be used.

The content of this parameter is only valid when setting a timer.

Establish count
INPUT; BINARY(4)
The number of times this timer will be established. Any value between 1 and 60 may be used. The

special value of -1 may be used to always have this timer established after it expires.

The content of this parameter is only valid when setting a timer.

Key length
INPUT; BINARY(4)
The key length when using a keyed data queue or user queue. Any value between 0 and 256 may be used, where 0 indicates the data queue or user queue is not keyed.

The content of this parameter is only valid when setting a timer.

Key value
INPUT; CHAR(256)
The key value when using a keyed data queue or user queue.

The content of this parameter is only valid when setting a timer.

User data
INPUT; CHAR(60)
The user data that is to be included in the timer-expired entry when the timer expires.

The content of this parameter is only valid when setting a timer.

Note: This data is treated as character data only and should not contain pointers.

Optional Parameter

Queue type
INPUT; CHAR(1)
The type of queue you specified for the queue name parameter.

D Data queue
U User queue

Return and Reason Codes

Figure 6-55. Return and Reason Codes for the QOLTIMER API

Return / Reason Code	Meaning	Recovery
0/0	Operation successful.	Continue processing.
81/9999	Internal system error detected. Escape message CPF91F0 will be sent to the application program when this return and reason code is received.	See messages in the job log for further information. Report the problem using the ANZPRB command.
82/1011	Queue type not valid.	Correct the queue type parameter. Try the request again.
83/1001	Key length not valid.	Correct the key length parameter. Try the request again.
83/1009	Timer operation not valid.	Correct the operation parameter. Try the request again.
83/1010	Timer interval not valid.	Correct the interval parameter. Try the request again.
83/1011	Number of times to establish timer not valid.	Correct the establish count parameter. Try the request again.
83/3400	Timer not valid on cancel operation.	Correct the timer to cancel parameter. Try the request again.
83/3401	All timers are currently set for the requested set operation.	Cancel a timer. Try the request again.
83/3402	Timer not set on cancel operation.	Continue processing.

Error Messages

CPF91F0 E Internal system error.

Chapter 7. Debugging of User-Defined Communications Applications

This section is intended to help you debug your user-defined communications applications. It contains information about:

- System services and tools
- Error codes reported to the application program and QSYSOPR operation
- Common error list

System Services and Tools

There are several tools on the AS/400 system you can use to debug your user-defined communications application. Some of the system provided tools that are useful for developing user-defined communications applications include the following CL commands:

- Program Debug (STRDBG)
- Work with Job, Work with Communications Status (WRKJOB OPTION(*CMNSTS))
- Work with Job, Display Job Log (WRKJOB OPTION(*JOBLOG))
- Display Connection Status (DSPCNNSTS)
- Display Inbound Routing Data (press F6 (Display inbound routing information) following the DSPCNNSTS command)
- Check Communications Trace (CHKCMNTRC)
- Delete Communications Trace (DLTCMNTRC)
- End Communications Trace (ENDCMNTRC)
- Print Communications Trace (PRTCMNTRC)
- Start Communications Trace (STRCMNTRC)
- Start System Service Tools (STRSST)
  - Work with communications trace
  - Work with error log
- Dump System Object (DMPSYSOBJ)

Program Debug

Program debug (STRDBG) allows you to trace the program and variables, set stops, change variables, and display variables. You can use this function to verify that the parameters are passed correctly.

Work with Communications Status

The Work with Job command, Work with Communications Status option, (WRKJOB OPTION(*CMNSTS)) shows the enabled links and operation counts for each link. It also reports information such as the communications handle the last operation requested, and the total input, output, and other operations requested. This information is shown for every link enabled by the job.

Display Job Log

The Work with Job command, selecting the Display job log option (WRKJOB OPTION(*JOBLOG)) allows you to view the messages in the job log that help determine the exact cause of the problem.

Display Connection Status

The Display Connection Status (DSPCNNSTS) command shows information about the switched virtual circuits (SVCs) and permanent virtual circuits (PVCs) that are in use by the application using the device description.

Note: The Display Line Description (DSPLIND) command also shows for each line, the SVCs that are in use, available, or attached to a controller description. This is not true for PVCs.

Display Inbound Routing Information

Pressing F6 (Display inbound routing information) when the Display Connection Status display is shown (DSPCNNSTS command) shows the results of the calls to the Set Filter (QOLSETF) API. It also helps to determine which device description has set a filter with duplicate inbound routing information.

Work with Communications Trace

Using the communications trace function you can obtain information about a communications line. You can access the communications trace function through the following CL commands:

- Check Communications Trace (CHKCMNTRC)
- Delete Communications Trace (DLTCMNTRC)
- End Communications Trace (ENDCMNTRC)
- Print Communications Trace (PRTCMNTRC)
- Start Communications Trace (STRCMNTRC)

For more information on using the communications trace CL commands, see the *Communications Management Guide*.

You can also access the communications trace function through the system service tools. You can use this function by entering the Start System Service Tools (STRSST) command and selecting the option to start a service tool.

Using the option to Work with communications trace shows data just as it appears to the network. If the application requests that data be sent and the request does not appear in the communications trace, the request is rejected. The return and reason codes, and the error code reported in the parameter list for the Send Data (QOLSEND) API indicates the reason the request was rejected.

Work with Error Log

The error log utility is part of the system service tools. You can use it by entering the Start System Service Tools (STRSST) command and selecting the option to start a service tool.

Some communications errors are reported by the system to the error log. A remote application that is communicating with a user-defined communications application on the local system, could cause an entry to be generated in the error log if one of the following conditions are met:

- When using a LAN, data is not received by the application and exceeds internal threshold values (3 Mb).
- When using an X.25 network, data is not received by the application and exceeds internal threshold values (128KB).

For both cases, the associated message in QSYSOPR identifies the error log that contains the error log entry. The error log entry contains information only.

Dump System Object to View User Spaces

The Dump System Object (DMPSYSOBJ) command is used to inspect the user spaces after they are filled in by your application. The following examples indicate what the user spaces look like for some of the operations.

User Space to Set a Filter to Route Inbound Data:

This user space is filled in to activate two X.25 filters which will route any X.25 call containing X'BB', or X'DD' in the first byte of call user data (protocol ID byte).

```
5738SS1 V2R1M0  910524                    AS/400 DUMP          006625/QSECOFR/QPADEV0001     12/21/90 12:42:07      PAGE    1
DMPSYSOBJ PARAMETERS
OBJ- OUTBUFFER                      CONTEXT-USRDFNCMN
OBJTYPE- *USRSPC
OBJECT TYPE-          SPACE                                        *USRSPC
NAME-        OUTBUFFER                    TYPE-          19   SUBTYPE-          34
LIBRARY-     USRDFNCMN                    TYPE-          04   SUBTYPE-          01
CREATION-    12/21/90  12:40:03           SIZE-          00002200
OWNER-       QSECOFR                      TYPE-          08   SUBTYPE-          01
ATTRIBUTES-          0800                 ADDRESS-       00A00A00   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934D6E4 E3C2E4C6   C6C5D940 40404040 40404040 40404040  *       -  OUTBUFFER            *
  000020   40404040 40404040 E0000000 00000000   00002000 00800000 00000000 00000000  *        \                      *
  000040   00000000 00000000 0005004D 42000400   00000000 00000000 00000000 00000000  *           (a                  *
SPACE-
  000000   01000002 001001BB 00000000 00000000   00000000 000001DD 00000000 00000000  *       Y               t      *
  000020   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                              *
        LINES  000040    TO    001FFF  SAME AS ABOVE
POINTERS-
  NONE
OIR DATA-
TEXT-
  000000   D8D7C1C4 C5E5F0F0 F0F1D8E2 C5C3D6C6   D9404040 F0F0F6F6 F2F5              *QPADEV0001QSECOFR   006625      *
SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1         *
  000020   40404040 40404040 404040E5 F2D9F1D4   F0F0F9F0 F1F2F2F1 F1F2F4F0 F0F44040  *          V2R1M00901221124004  *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040  *                               *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                               *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 40400000 00000000  *                               *
  0000A0   00000000 00000000                                                          *                               *
END OF DUMP
       *****  E N D  O F  L I S T I N G  *****
```

Figure 7-1. User Space to Set a Filter to Route Incoming X.25 Calls

User Space for X'B000' Operation, Initiating an SVC Call: The user space below has been filled in to initiate an SVC call specifying the following:

- Default packet and window sizes
- D-bit (not selected)
- Reverse charging (not selected)
- Fast select (not selected)
- Closed user group (not selected)
- Other facilities (not selected)
- One byte of call user data, X'BB', which is the protocol identifier
- X.25 reset not supported by the user-defined communications application program
- 16KB is the maximum amount of contiguous data to be received
- Automatic flow control value of 32

```
5738SS1 V2R1M0  910524                    AS/400 DUMP          006625/QSECOFR/QPADEV0001     12/21/90 12:47:42        PAGE    1
DMPSYSOBJ PARAMETERS
OBJ- OUTPUTBUF                        CONTEXT-USRDFNCMN
OBJTYPE- *USRSPC
OBJECT TYPE-          SPACE                                          *USRSPC
NAME-        OUTPUTBUF                      TYPE-          19   SUBTYPE-          34
LIBRARY-     USRDFNCMN                      TYPE-          04   SUBTYPE-          01
CREATION-    12/21/90  12:36:28             SIZE-          00001200
OWNER-       QSECOFR                        TYPE-          08   SUBTYPE-          01
ATTRIBUTES-          0800                   ADDRESS-       00A00100   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934D6E4 E3D7E4E3   C2E4C640 40404040 40404040 40404040  *       - OUTPUTBUF             *
  000020   40404040 40404040 E0000000 00000000   00001000 00800000 00000000 00000000  *        \                      *
  000040   00000000 00000000 0005004D 42000400   00000000 00000000 00000000 00000000  *           (a                  *
SPACE-
  000000   02000000 FFFFFFFF FFFFFFFF 00000000   00000008 40100001 00000000 00000000  *                               *
  000020   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
        LINES  000040    TO    0000BF  SAME AS ABOVE
  0000C0   00000000 00000000 00000000 00000000   00000000 00000001 BB000000 00000000  *                        Y      *
  0000E0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
        LINES  000100    TO    0001BF  SAME AS ABOVE
  0001C0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00004000  *                               *
  0001E0   00200000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
  000200   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
        LINES  000220    TO    000FFF  SAME AS ABOVE
POINTERS-
  NONE
OIR DATA-
TEXT-
  000000   D8D7C1C4 C5E5F0F0 F0F2D8E2 C5C3D6C6   D9404040 F0F0F6F6 F2F7              *QPADEV0002QSECOFR   006627      *
SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1          *
  000020   40404040 40404040 404040E5 F2D9F1D4   F0F0F9F0 F1F2F2F1 F1F2F3F6 F2F84040  *          V2R1M00901221123628  *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040  *                               *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                               *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 40400000 00000000  *                               *
  0000A0   00000000 00000000                                                          *                               *
END OF DUMP
                                        * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 7-2. User Space to Send an SVC Call

User Space Containing an Incoming X.25 Call, Operation X'B201': This user space shows the following:

- The call is using SVC 005
- Both transmit and receive packet sizes are 128
- Both transmit and receive window sizes are 7
- The calling DTE address is 40100000
- No other facilities are requested
- One byte of call user data, X'BB', which is the protocol identifier

The application received this call because it had set a filter to indicate to the system that it should route incoming X.25 calls that have the first byte of call user data (the protocol identifier) equal to X'BB' to the application.

```
5738SS1 V2R1M0  910524                    AS/400 DUMP          006625/QSECOFR/QPADEV0001     12/21/90 12:47:55      PAGE    1
DMPSYSOBJ PARAMETERS
OBJ- INBUFFER                        CONTEXT-USRDFNCMN
OBJTYPE- *USRSPC
OBJECT TYPE-          SPACE                                        *USRSPC
NAME-        INBUFFER                      TYPE-          19   SUBTYPE-          34
LIBRARY-     USRDFNCMN                     TYPE-          04   SUBTYPE-          01
CREATION-    12/21/90  12:40:03            SIZE-          00002200
OWNER-       QSECOFR                       TYPE-          08   SUBTYPE-          01
ATTRIBUTES-          0800                  ADDRESS-       00A00400   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934C9D5 C2E4C6C6   C5D94040 40404040 40404040 40404040  *       -  INBUFFER             *
  000020   40404040 40404040 E0000000 00000000   00002000 00800000 00000000 00000000  *        \                      *
  000040   00000000 00000000 0005004D 42000400   00000000 00000000 00000000 00000000  *           (a                  *
SPACE-
  000000   00000005 00800007 00800007 00000000   00000008 40100000 00000000 00000000  *                               *
  000020   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
       LINES  000040    TO    0000BF  SAME AS ABOVE
  0000C0   00000000 00000000 00000000 00000000   00000000 00000001 BB000000 00000000  *                        Y      *
  0000E0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
       LINES  000100    TO    001FFF  SAME AS ABOVE
POINTERS-
  NONE
OIR DATA-
TEXT-
  000000   D8D7C1C4 C5E5F0F0 F0F1D8E2 C5C3D6C6   D9404040 F0F0F6F6 F2F5              *QPADEV0001QSECOFR   006625      *
SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1          *
  000020   40404040 40404040 404040E5 F2D9F1D4   F0F0F9F0 F1F2F2F1 F1F2F4F0 F0F34040  *          V2R1M00901221124003  *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040  *                               *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                               *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 40400000 00000000  *                               *
  0000A0   00000000 00000000                                                          *                               *
END OF DUMP
                                        * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 7-3. User Space Containing an Incoming X.25 Call

User Space to Accept an Incoming X.25 Call, Operation X'B400': This user space was filled in to accept the incoming call, request default packet and window sizes, and no other additional facilities. The a maximum amount of contiguous data is set at 16KB and the automatic flow control is set at 32.

```
5738SS1 V2R1M0  910524                    AS/400 DUMP          006625/QSECOFR/QPADEV0001     12/21/90 12:48:06       PAGE    1
DMPSYSOBJ PARAMETERS
OBJ- OUTBUFFER                        CONTEXT-USRDFNCMN
OBJTYPE- *USRSPC
OBJECT TYPE-           SPACE                                          *USRSPC
NAME-        OUTBUFFER                      TYPE-          19   SUBTYPE-          34
LIBRARY-     USRDFNCMN                      TYPE-          04   SUBTYPE-          01
CREATION-    12/21/90  12:40:03             SIZE-          00002200
OWNER-       QSECOFR                        TYPE-          08   SUBTYPE-          01
ATTRIBUTES-          0800                   ADDRESS-       00A00A00   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934D6E4 E3C2E4C6   C6C5D940 40404040 40404040 40404040  *       -  OUTBUFFER            *
  000020   40404040 40404040 E0000000 00000000   00002000 00800000 00000000 00000000  *        \                      *
  000040   00000000 00000000 0005004D 42000400   00000000 00000000 00000000 00000000  *           (a                  *
SPACE-
  000000   00000000 FFFFFFFF FFFFFFFF 00000000   00000000 00000000 00000000 00000000  *                               *
  000020   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
       LINES  000040    TO    0001BF  SAME AS ABOVE
  0001C0   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00004000  *                               *
  0001E0   00200000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
  000200   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
       LINES  000220    TO    001FFF  SAME AS ABOVE
POINTERS-
  NONE
OIR DATA-
TEXT-
  000000   D8D7C1C4 C5E5F0F0 F0F1D8E2 C5C3D6C6   D9404040 F0F0F6F6 F2F5              *QPADEV0001QSECOFR   006625      *
SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1          *
  000020   40404040 40404040 404040E5 F2D9F1D4   F0F0F9F0 F1F2F2F1 F1F2F4F0 F0F44040  *          V2R1M00901221124004  *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040  *                               *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                               *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 40400000 00000000  *                               *
  0000A0   00000000 00000000                                                          *                               *
END OF DUMP
                                        * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 7-4. User Space to Accept an Incoming X.25 Call

User Spaces for Sending Data, Operation X'0000':
Two user spaces are shown below. The first is the output buffer and the second is the output buffer descriptor.

The user spaces below are filled in to send three data units of 512 bytes each. The first two data units have the more data indicator turned on, indicating that all the data units are contiguous.

Note: This link was enabled, specifying a data unit size of 512 bytes.

```
5738SS1 V2R1M0  910524                    AS/400 DUMP          006625/QSECOFR/QPADEV0001     12/21/90 12:55:19      PAGE    1
DMPSYSOBJ PARAMETERS
OBJ- OUTPUTBUF                       CONTEXT-USRDFNCMN
OBJTYPE- *USRSPC
OBJECT TYPE-          SPACE                                        *USRSPC
NAME-        OUTPUTBUF                     TYPE-          19   SUBTYPE-          34
LIBRARY-     USRDFNCMN                     TYPE-          04   SUBTYPE-          01
CREATION-    12/21/90  12:36:28            SIZE-          00001200
OWNER-       QSECOFR                       TYPE-          08   SUBTYPE-          01
ATTRIBUTES-          0800                  ADDRESS-       00A00100   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934D6E4 E3D7E4E3   C2E4C640 40404040 40404040 40404040  *       -  OUTPUTBUF            *
  000020   40404040 40404040 E0000000 00000000   00001000 00800000 00000000 00000000  *        \                      *
  000040   00000000 00000000 0005004D 42000400   00000000 00000000 00000000 00000000  *          (a                   *
SPACE-
  000000   F0F10000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *01                             *
  000020   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
       LINES  000040    TO    0001FF  SAME AS ABOVE
  000200   F0F20000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *02                             *
  000220   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
       LINES  000240    TO    0003FF  SAME AS ABOVE
  000400   F0F30000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *03                             *
  000420   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
       LINES  000440    TO    000FFF  SAME AS ABOVE
POINTERS-
  NONE
OIR DATA-
TEXT-
  000000   D8D7C1C4 C5E5F0F0 F0F2D8E2 C5C3D6C6   D9404040 F0F0F6F6 F2F7              *QPADEV0002QSECOFR   006627      *
SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1          *
  000020   40404040 40404040 404040E5 F2D9F1D4   F0F0F9F0 F1F2F2F1 F1F2F3F6 F2F84040  *          V2R1M00901221123628  *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040  *                               *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                               *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 40400000 00000000  *                               *
  0000A0   00000000 00000000                                                          *                               *
END OF DUMP
                                        * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 7-5. User Space (Buffer) to Send Three Data Units

```
5738SS1 V2R1M0  910524                    AS/400 DUMP          006625/QSECOFR/QPADEV0001     12/21/90 12:55:58      PAGE    1
DMPSYSOBJ PARAMETERS
OBJ- OUTPUTBUFD                      CONTEXT-USRDFNCMN
OBJTYPE- *USRSPC
OBJECT TYPE-          SPACE                                        *USRSPC
NAME-        OUTPUTBUFD                    TYPE-          19   SUBTYPE-          34
LIBRARY-     USRDFNCMN                     TYPE-          04   SUBTYPE-          01
CREATION-    12/21/90  12:36:27            SIZE-          00000400
OWNER-       QSECOFR                       TYPE-          08   SUBTYPE-          01
ATTRIBUTES-          0800                  ADDRESS-       009FFE00   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934D6E4 E3D7E4E3   C2E4C6C4 40404040 40404040 40404040  *       -  OUTPUTBUFD           *
  000020   40404040 40404040 E0000000 00000000   00000200 00800000 00000000 00000000  *        \                      *
  000040   00000000 00000000 0005004D 42000400   00000000 00000000 00000000 00000000  *          (a                   *
SPACE-
  000000   02000100 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
  000020   02000100 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
  000040   02000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
  000060   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                               *
       LINES  000080    TO    0001FF  SAME AS ABOVE
POINTERS-
  NONE
OIR DATA-
TEXT-
  000000   D8D7C1C4 C5E5F0F0 F0F2D8E2 C5C3D6C6   D9404040 F0F0F6F6 F2F7              *QPADEV0002QSECOFR   006627      *
SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1          *
  000020   40404040 40404040 404040E5 F2D9F1D4   F0F0F9F0 F1F2F2F1 F1F2F3F6 F2F74040  *          V2R1M00901221123627  *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040  *                               *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                               *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 40400000 00000000  *                               *
  0000A0   00000000 00000000                                                          *                               *
END OF DUMP
                                        * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 7-6. User Space (Descriptor Element) to Describe the Three Data Units

User Spaces for Receiving Data, Operation X'0001': Two user spaces are shown below. The first is the input buffer and the second is the input buffer descriptor.

The user spaces below are filled in showing that 2 data units were received. The first data unit has the more data indicator turned on in the buffer descriptor for the data unit. This means that the X.25 more indicator was turned on in all the X.25 packets that this data unit contains. The second data unit does not have the more data indicator turned on, indicating that the last X.25 packet in the data unit had the X.25 more indicator turned off. The first and second data unit are considered to be logically contiguous to the application program.

Note: This link was enabled specifying a data unit size of 1024 bytes. The sending system sent the data in data unit sizes of 512 bytes and they were combined into the 1024 byte data unit size by the local system. The data unit size is not negotiated end-to-end, neither is the maximum amount of contiguous data or the automatic flow control. Because the values are important, each application should be aware of what the other application has specified for each value. Refer to "Sending and Receiving Data Packets" on page 4-25 for more information.

```
5738SS1 V2R1M0  910524                    AS/400 DUMP          006625/QSECOFR/QPADEV0001     12/21/90 12:59:33      PAGE    1
DMPSYSOBJ PARAMETERS
OBJ- INBUFFER                         CONTEXT-USRDFNCMN
OBJTYPE- *USRSPC
OBJECT TYPE-          SPACE                                          *USRSPC
NAME-        INBUFFER                        TYPE-          19   SUBTYPE-          34
LIBRARY-     USRDFNCMN                       TYPE-          04   SUBTYPE-          01
CREATION-    12/21/90  12:40:03              SIZE-          00002200
OWNER-       QSECOFR                         TYPE-          08   SUBTYPE-          01
ATTRIBUTES-          0800                    ADDRESS-       00A00400   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934C9D5 C2E4C6C6   C5D94040 40404040 40404040 40404040  *       - INBUFFER             *
  000020   40404040 40404040 E0000000 00000000   00002000 00800000 00000000 00000000  *        \                     *
  000040   00000000 00000000 0005004D 42000400   00000000 00000000 00000000 00000000  *          (a                  *
SPACE-
  000000   F0F10000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *01                           *
  000020   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                             *
       LINES  000040    TO    0001FF  SAME AS ABOVE
  000200   F0F20000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *02                           *
  000220   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                             *
       LINES  000240    TO    0003FF  SAME AS ABOVE
  000400   F0F30000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *03                           *
  000420   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                             *
       LINES  000440    TO    001FFF  SAME AS ABOVE
POINTERS-
  NONE
OIR DATA-
TEXT-
  000000   D8D7C1C4 C5E5F0F0 F0F1D8E2 C5C3D6C6   D9404040 F0F0F6F6 F2F5              *QPADEV0001QSECOFR   006625      *
SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1        *
  000020   40404040 40404040 404040E5 F2D9F1D4   F0F0F9F0 F1F2F2F1 F1F2F4F0 F0F34040  *          V2R1M00901221124003  *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040  *                               *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                               *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 40400000 00000000  *                               *
  0000A0   00000000 00000000                                                          *                               *
END OF DUMP
                                        * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 7-7. User Space (Buffer) Containing the Three Data Units

```
5738SS1 V2R1M0  910524                    AS/400 DUMP          006625/QSECOFR/QPADEV0001     12/21/90 12:59:41       PAGE    1
DMPSYSOBJ PARAMETERS
OBJ- INBUFFERD                       CONTEXT-USRDFNCMN
OBJTYPE- *USRSPC
OBJECT TYPE-          SPACE                                        *USRSPC
NAME-       INBUFFERD                      TYPE-          19   SUBTYPE-          34
LIBRARY-    USRDFNCMN                      TYPE-          04   SUBTYPE-          01
CREATION-   12/21/90  12:40:03             SIZE-          00000400
OWNER-      QSECOFR                        TYPE-          08   SUBTYPE-          01
ATTRIBUTES-         0800                   ADDRESS-       00A00200   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934C9D5 C2E4C6C6   C5D9C440 40404040 40404040 40404040  *       - INBUFFERD            *
  000020   40404040 40404040 E0000000 00000000   00000200 00800000 00000000 00000000  *        \                     *
  000040   00000000 00000000 0005004D 42000400   00000000 00000000 00000000 00000000  *           (a                 *
SPACE-
  000000   04000100 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                              *
  000020   02000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                              *
  000040   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                              *
       LINES  000060    TO    0001FF  SAME AS ABOVE
POINTERS-
  NONE
OIR DATA-
TEXT-
  000000   D8D7C1C4 C5E5F0F0 F0F1D8E2 C5C3D6C6   D9404040 F0F0F6F6 F2F5              *QPADEV0001QSECOFR   006625      *
SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1         *
  000020   40404040 40404040 404040E5 F2D9F1D4   F0F0F9F0 F1F2F2F1 F1F2F4F0 F0F34040  *          V2R1M00901221124003 *
  000040   40404040 40404040 4040404u 40404040   40404040 40404040 40404040 40404040  *                              *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                              *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 40400000 00000000  *                              *
  0000A0   00000000 00000000                                                         *                              *
END OF DUMP
                                        * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 7-8. User Space (Descriptor Element) Describing the Three Data Units

User Space to Clear a Connection or Call, Operation

X'B100': This user space was filled in to end an SVC connection or clear an incoming call. No facilities or clear user data are requested with this, but cause and diagnostic codes are specified (these are not ISO or SNA codes).

```
5738SS1 V2R1M0  910524                    AS/400 DUMP          006625/QSECOFR/QPADEV0001     12/21/90 13:14:48       PAGE    1
DMPSYSOBJ PARAMETERS
OBJ- OUTBUFFER                       CONTEXT-USRDFNCMN
OBJTYPE- *USRSPC
OBJECT TYPE-          SPACE                                        *USRSPC
NAME-       OUTBUFFER                      TYPE-          19   SUBTYPE-          34
LIBRARY-    USRDFNCMN                      TYPE-          04   SUBTYPE-          01
CREATION-   12/21/90  12:40:03             SIZE-          00002200
OWNER-      QSECOFR                        TYPE-          08   SUBTYPE-          01
ATTRIBUTES-         0800                   ADDRESS-       00A00A00   0000
SPACE ATTRIBUTES-
  000000   00000080 00000060 1934D6E4 E3C2E4C6   C6C5D940 40404040 40404040 40404040  *       - OUTBUFFER            *
  000020   40404040 40404040 E0000000 00000000   00002000 00800000 00000000 00000000  *        \                     *
  000040   00000000 00000000 0005004D 42000400   00000000 00000000 00000000 00000000  *           (a                 *
SPACE-
  000000   0000BEBE 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *  XX                          *
  000020   00000000 00000000 00000000 00000000   00000000 00000000 00000000 00000000  *                              *
       LINES  000040    TO    001FFF  SAME AS ABOVE
POINTERS-
  NONE
OIR DATA-
TEXT-
  000000   D8D7C1C4 C5E5F0F0 F0F1D8E2 C5C3D6C6   D9404040 F0F0F6F6 F2F5              *QPADEV0001QSECOFR   006625      *
SERVICE-
  000000   40404040 40404040 40404040 40404040   40404040 40F14040 40404040 40404040  *                    1         *
  000020   40404040 40404040 404040E5 F2D9F1D4   F0F0F9F0 F1F2F2F1 F1F2F4F0 F0F44040  *          V2R1M00901221124004 *
  000040   40404040 40404040 40404040 40404040   40404040 40404040 40404040 40404040  *                              *
  000060   40404040 40404040 40404040 40404040   40404040 40404040 00000000 00000000  *                              *
  000080   00000000 00000000 00000000 00000000   00000000 00000000 40400000 00000000  *                              *
  0000A0   00000000 00000000                                                         *                              *
END OF DUMP
                                        * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Figure 7-9. User Space to Send an SVC Clear

Error Codes

The system and user-defined communications support reports important information that is useful for determining recovery actions when an error occurs. This information is referred to as error codes that are reported either to the job log or to the QSYSOPR message queue. For a complete list of the messages that are signaled by the user-defined communications APIs, see "Messages" on page 4-29.

In some cases error codes are reported to your application in the error specific parameter. The following sections list the valid error codes. Some of the error codes represent actual coding errors, others only report additional information. For information about the error codes for the individual user-defined communications APIs, see Chapter 6, "User-Defined Communications Support APIs" on page 6-1.

Local Area Network (LAN) Error Codes

Figure 7-10 shows the valid hexadecimal codes your application can receive as a result of a call to the QOLSEND API using operation code X'0000'. The codes indicate that the data was never sent on the line. Associated with these error codes is a message in QSYSOPR, indicating the device description that caused the error, and the error code. After receiving the error code, the link will still be enabled and usable.

These error codes indicate to your application that a coding error was made and should be corrected.

Figure 7-10. Error Codes Received While Sending Data over LAN

Error Code	Description	Cause
3300 2A55	Routing length not valid	Routing length is not valid, or length does not equal length in routing field.
3300 2A5D	Maximum frame size limit exceeded	Length of data is greater than maximum frame size supported by the source SAP
5300 2A7B	Access Control not valid	Access Control specified is not supported
3300 2AA9	SAP address not valid	SAP address is not configured in the line description
3300 2AA9	SAP address not valid	SAP address is not configured in the line description
3300 2AD4	Data length too small (Ethernet Version 2 only)	Data must be at least 48 bytes long (46 bytes of data, plus 2 bytes for the Ethernet type field)
3300 2AD5	Ethernet type field is not valid (Ethernet Version 2 only)	Ethernet type field (first two bytes of data)

X.25 Error Codes

Figure 7-11 shows the valid error codes your application can receive as a result of

- A call to the QOLSEND API with operation X'B400' to accept an SVC call
- A call to the Receive Data (QOLRECV) API which returns the results of the open connection request operation, X'B101'
- The connection failure indication, reported by operation X'B301'

These error codes indicate to your application that a coding error was made, or a failure condition occurred.

Figure 7-11 (Page 1 of 2). Error Codes Reported on X'B001', X'B301', and X'B400' Operations

Error Code	Description	Cause
1200 3122	Outgoing channel not available	The logical channel is still active and in the process of being deactivated
3200 3050	Restart in progress	Temporary condition: retry operation
3200 3172	Outgoing channel not available	Temporary condition: retry operation
3200 3368	Remote address length not valid	Remote address length not supported by the network
3200 3384	Facility field error	A facility was encoded incorrectly or a duplicate facility was encoded
3200 3388	Facility field too long	The total length of the facilities, which includes user-specified facilities, the NUI facility from the line description, and system generated facilities, exceeded X.25 limits (109 bytes)
3200 338C	Response restricted by fast select	User data is not allowed with restriction

Figure 7-11 (Page 2 of 2). Error Codes Reported on X'B001', X'B301', and X'B400' Operations

Error Code	Description	Cause
3200 3394	User data not allowed	User data is not allowed on the call accept if fast select was not requested.
3200 33CC	Call user data length not valid	The length of call user data is greater than 16 and fast select is not selected.
4200 3210	Reset request transmitted	The virtual circuit was reset by the local system. Refer to cause and diagnostic codes to determine recovery.
4200 3220	Clear request transmitted	The virtual circuit was cleared by the local system. Refer to cause and diagnostic codes to determine recovery.
4200 3222	Clear request transmitted	The virtual circuit was cleared by the local system because there was a problem with the packet size in the call accept. This is either a configuration problem or a network problem. Verify that the default packet size in the line description is correct.
4200 3224	Clear request transmitted	The virtual circuit was cleared by the local system because there was a problem with the window size in the call accept. This is either a configuration problem or a network problem. Verify that the default window size in the line description is correct.
4200 3230	Restart request transmitted	The virtual circuit was cleared by the local system. Refer to cause and diagnostic codes for more information.
4200 3280	Time-out on call	Call timed out
4600 3134	Clear indication was received	The virtual circuit was cleared by either the remote system or the network. Refer to cause and diagnostic codes for more information.
4600 3138	Restart indication received	Temporary condition; refer to the cause and diagnostic codes reported to correct the problem, then retry the operation

Figure 7-12 shows the valid error codes your application can receive as a result of a call to the QOLRECV API with an operation code returned as X'B101'.

These error codes indicate to your application that the connection was cleared or reset for the following reasons.

Figure 7-12. Error Codes Reported on the X'B101' Operation

Error Code	Description	Cause
3200 3388	Facility field too long	The total length of the facilities, which includes user-specified facilities, the NUI facility from the line description, and system generated facilities, exceeded X.25 limits (109 bytes)
3200 3394	User data not allowed	User data is not allowed when fast select is not selected.
3200 33CC	Call user data length not valid	The length of call user data is greater than 16 and fast select is not selected.
4200 3240	Time-out on reset	The clear request resulted in an X.25 reset, which timed out
4200 3284	Time-out on clear	The remote system did not respond to the CLEAR within the time-out value
4600 3134	Clear indication was received	The virtual circuit was cleared by either the remote system or the network. Refer to cause and diagnostic codes for more information.

Figure 7-13 on page 7-11 shows the valid error codes your application can receive as a result of a call to the QOLRECV API, returning the operation code, X'BF01'.

These error codes indicate to your application that the connection was cleared or reset for the following reasons.

Figure 7-13. Error Codes Reported on the X'BF01' Operation

Error Code	Description	Cause
3200 3050	Network Restart in progress	Temporary condition; connection is no longer active.
3200 3A0C	Close pending	The virtual circuit is being closed.
3200 3A0D	Reset pending	The virtual circuit is in the process of being reset by either the remote system or the network.
4200 3210	Reset packet transmitted	A Reset packet was transmitted from the local system.
4200 3240	Time-out on reset	The clear request resulted in an X.25 reset, which timed out
4600 3130	Reset indication was received	The virtual circuit received a reset by either the remote system or the network. Refer to cause and diagnostic codes for more information.
4600 3134	Clear indication was received	The virtual circuit was cleared by either the remote system or the network. Refer to cause and diagnostic codes for more information.

Figure 7-14 shows the valid error codes your application can receive as a result of a call to the QOLSEND API with an operation code returned as X'0000'.

These error codes indicate to your application that the connection was cleared or reset for the following reasons.

Figure 7-14. Error Codes Resulting from a X'0000' Operation

Error Code	Description	Cause
3200 3050	Network restart in progress	Temporary condition; connection is no longer active.
3200 336A	Q/M bit sequence not valid	If the data is qualified, the Q bit must be set for all data units.
3200 33C8	Data length not valid	The length of the packet is not supported for this virtual circuit.
3200 3A0C	Close pending	The virtual circuit is being closed.
3200 3A0D	Reset pending	The virtual circuit is in the process of being reset by either the remote system or the network.
4200 3284	Interrupt timed out	The local DTE sent an interrupt packet. The response to this packet was not received within the time-out period, and the connection has been reset by the AS/400 system.
4600 3130	Reset indication was received	The virtual circuit received a reset by either the remote system or the network. Refer to cause and diagnostic codes for more information.
4600 3134	Clear indication was received	The virtual circuit was cleared by either the remote system or the network. Refer to cause and diagnostic codes for more information.

Common Errors and Messages

This section shows some of the common errors that you or your application programmer may encounter. Some of these errors are detected by the APIs and reported to the application by the unsuccessful return and reason codes returned on each API. Other errors are program design errors, that your application programmer must detect and correct. The errors are listed by category:

Parameter errors:

- Switching use of connection identifiers (PCEP and UCEP)
- Switching use of timer handles
- Not encoding parameters if not used
- Operation code not in hexadecimal format
- Parameter not declared with proper length

User Space errors:

- Not encoding reserved space for fields not used
- Not initializing user space fields as necessary.

  The output user spaces can only be changed by the user-defined communications application. Operations are validated on each request. If there are fields that the current operation does not use, they should be set to contain zeros with X'00', to prevent a template error resulting from information on the previous operation still being in the user space. Not resetting the indicators in the output buffer descriptors on each operation and not zeroing out fields before making a call request may result in template errors.

Queue errors:

- Queue not created
- Queue created with different key length than specified in the parameter list of the Enable Link (QOLELINK) API

Receive Data (QOLRECV) API Errors:

- Not checking the more data output parameter and issuing another call to the QOLRECV API
- Not calling the QOLSETF API to set the filter to route inbound data to the application
- Using the wrong data unit descriptor for the data unit (each data unit has its own descriptor)

Send Data (QOLSEND) API Errors:

- After a call to the QOLSEND API with an operation code of X'B000', X'B100', or X'BF00', the application should then call the QRCVDTAQ API and wait for incoming data to be placed on the queues. The success or failure of these operations is reported through the QOLRECV API with operation codes of X'B001', X'B101' and X'BF01', respectively.
- Using the wrong data unit descriptor for the data unit (each data unit has its own descriptor)

Enable Link (QOLELINK) API Errors:

- User space names not unique
- Queue not created before program call
- Line description not created or incorrect prior to program call

Query Line Description (QOLQLIND) API Errors:

- Parameter buffer not large enough

Chapter 8. Data Stream Translation APIs

This section describes the data stream translation application program interfaces (APIs) that allow your user-written applications access to the data stream translation routines for 5250, 3270, and formatted buffer display data streams. Only display device data streams are supported by this API. For more information on display data streams using formatted buffers, see the *SNA Upline Facility Programmer's Guide*. The data stream translation APIs are listed in alphabetical order following a brief discussion of the function they provide and the considerations for using them in a user-written application program.

Using the data stream APIs, your applications can:

- Translate from a 3270 output data stream to a formatted buffer
- Translate from a 3270 output data stream to a 5250 data stream
- Translate from a formatted output buffer to a 5250 data stream
- Translate from a formatted input buffer to a 3270 data stream
- Translate from a 5250 input data stream to a formatted buffer
- Translate from a 5250 input data stream to a 3270 data stream

The following figures show the translation options available when your application calls the data stream translation APIs.

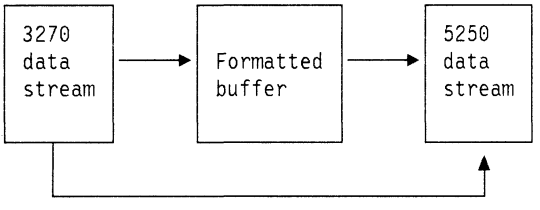


Figure 8-1. Translations for Output Operations

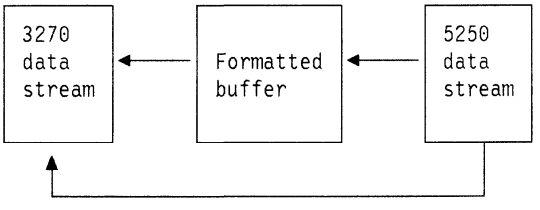


Figure 8-2. Translations for Input Operations

Using the Data Stream Translation APIs

The data stream APIs consist of three program calls that are shown in the following list:

End Data Stream Translation Session (QD0ENDTS) ends a session for data stream translation.
Start Data Stream Translation Session (QD0STRTS) starts a session for data stream translation.
Translate Data Stream (QD0TRNDS) translates a data stream in one format to another format.

When your application calls the QD0STRTS API, a translation session is opened using a user-specified device as a basis for the translation parameters. You can open as many sessions as you need, because for every session a unique session name is passed back to your application.

A call to the QD0TRNDS API does the actual data stream translation using the specified parameters to indicate the type of translation. Multiple translation sessions can be active at the same time. A translation session remains open, that is the handle remains valid, until the QD0ENDTS API is called using that handle or the job that called QD0STRTS ends. The final call to the QD0ENDTS API closes or ends the translation session.

Note: If you are using the same translation parameters for many translations, you may decide to use only one QD0STRTS call for each unique set of parameters to enhance performance.

Programming Restrictions

The 5250 data streams generated by the QD0TRNDS API for your application have the following restrictions:

- Read commands are not added to the end of a data stream. Your application is responsible for sending Read modified data tag (MDT) fields to the destination display.
- If the device for which the data stream is intended does not support data in row 1, column 1 then this location is restricted from use in the input field.
- The number of input fields is dependent on the type of work station controller. The following is a list of the devices and the maximum number of input and output fields allowed:

3270 display station	126
5250 local display station	255
5250 pass-through	126
5251 display station	126
5294 Remote Control Unit	230
5394 Remote Control Unit	255
PC Support/400 running work station function	254

- Fields that are detectable by light pens are not supported.

There are some 3270 data stream commands, orders, and attributes that are not supported. For a list of the 3270 data stream commands, orders, and attributes that are supported, see the *3270 Device Emulation Guide*.

All parameter values must be uppercased and left justified.

End Data Stream Translation Session (QD0ENDTS) API

Parameters

Required Parameter Group:

1	Translation session handle	Output	Char(16)
2	Error code	I/O	Char(*)

The End Data Stream Translation Session (QD0ENDTS) API ends a session for data stream translation.

Required Parameter Group

Translation session handle
OUTPUT; CHAR(16)
The name of the translation session. This name is returned to your application following the call to the QD0STRTS API.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF5D58 E Translation session handle parameter value not valid.
CPF5D67 E Severe error occurred while addressing parameter list.

Start Data Stream Translation Session (QD0STRTS) API

Parameters

Required Parameter Group:

1	Translation session handle	Output	Char(16)
2	Display device name	Input	Char(10)
3	Default screen size	Input	Char(10)
4	Alternate screen size	Input	Char(10)
5	Error code	I/O	Char(*)

The Start Data Stream Translation Session (QD0STRTS) API initiates a session for data stream translation. Your application can start as many translation sessions as you need.

Authorities and Locks

Device Authority
The user must have at least *USE authority to the device specified in the display device name parameter.

Required Parameter Group

Translation session handle
OUTPUT; CHAR(16)
The name of the translation session. This name is supplied to your application so that you can keep track of a particular session. It is also required that you pass this name to the other data stream APIs.

Display device name
INPUT; CHAR(10)
The name of the 5250 device for which the translation is being done. The 5250 data stream that is generated depends on the capabilities of the display device. You can specify the following values:

Name The name of a display device that is known to the system.

Note: An error will occur if the job you are using for data stream translation is not authorized to the device you specify.

***REQUESTER**
The display device that is associated with this job is to be used.

Note: An error will occur if there is no display device associated with this job. For example, the job is a batch job.

***BASIC**
The display device is assumed to have the lowest common characteristics. The following characteristics are assumed:

- The display is monochrome.
- The display has a screen size of 24x80. If a larger screen size is specified when *BASIC is specified for the display device name, an error occurs.
- Input in row 1, column 1 is not supported.
- The Home key does not work like the 3270 home key.
- The maximum number of input fields is 126.
- The language is defaulted to the Keyboard Type (QKBDTYPE) system value.

Default screen size
INPUT; CHAR(10)
The size of the screen for the selected display device. Either this value or the alternate screen size value is used depending on the command used in the 3270 data stream. The possible screen sizes are:

024X080 24 lines by 80 columns
027X132 27 lines by 132 columns

*DEVMAX The maximum screen size allowed by the device

Alternate screen size
INPUT; CHAR(10)
The alternate size of the screen for the selected display device. Either this value or the default screen size value is used depending on the command used in the 3270 data stream. The possible screen sizes are:

024X080 24 lines by 80 columns
027X132 27 lines by 132 columns
*DEVMAX The maximum screen size allowed by the device

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF5D50 E Display device description &1 not found.
CPF5D51 E Device &1 is not a display device.
CPF5D52 E Not authorized to display device &1.
CPF5D5B E Value &1 for default screen size parameter not valid.
CPF5D61 E Value for display device parameter not valid.
CPF5D66 E Value for alternate screen size parameter not valid.
CPF5D67 E Severe error occurred while addressing parameter list.
CPF5D68 E Default screen size parameter is not valid.
CPF5D69 E Alternate screen size parameter is not valid.

Translate Data Stream (QD0TRNDS) API

Parameters

Required Parameter Group:

1	Translation session handle	Input	Char(16)
2	To buffer	Output	Char(*)
3	To buffer output length	Output	Binary(4)
4	To buffer length	Input	Binary(4)
5	To buffer type	Input	Char(10)
6	From buffer	Input	Char(*)
7	From buffer length	Input	Binary(4)
8	From buffer type	Input	Char(10)
9	Operation	Input	Char(1)
10	Error code	I/O	Char(*)

The Translate Data Stream (QD0TRNDS) API translates data on one format to another format. The data formats depend on the parameter values you specify.

Required Parameter Group

Translation session handle
INPUT; CHAR(16)
The name of the translation session. This name is returned to your application following the call to the QD0STRTS API.

To buffer
OUTPUT; CHAR(*)
The buffer used to contain the output of the data stream translation. This value should be large enough to contain the expected results.

To buffer output length
OUTPUT; BINARY(4)
The length of the translated data that is placed in the to buffer parameter.

To buffer length
INPUT; BINARY(4)
The length of the buffer that is available for output.

To buffer type
INPUT; CHAR(10)
The type of data to be put into the to buffer parameter. The possible values are:

5250 Create a 5250 data stream
3270 Create a 3270 data stream
*FORMAT Create a formatted buffer for the data.

See Figure 8-3 on page 8-4 for a list of the allowable combinations of this parameter with the operations and from buffer type parameters.

From buffer
INPUT; CHAR(*)
The buffer that contains the data to be translated.

From buffer length
INPUT; BINARY(4)
The length of the data contained in the from buffer parameter.

From buffer type
INPUT; CHAR(10)
The type of data that is contained in the from buffer parameter. The possible values are:

5250 Contains a 5250 data stream
3270 Contains a 3270 data stream
*FORMAT Contains a formatted buffer for the data.

See Figure 8-3 on page 8-4 for a list of the allowable combinations of this parameter with the operations and to buffer type parameters.

Operation
INPUT; CHAR(1)
Indicates whether the data to be translated is input or output data. You can specify the following values:

I The data to be translated is for an input operation
O The data to be translated is for an output operation

See Figure 8-3 on page 8-4 for a list of the allowable combinations of this parameter with the to buffer type and from buffer type parameters.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

The following table lists the valid combinations of the from buffer type, to buffer type, and operations parameters.

Figure 8-3. Valid Parameter Combinations

Operation	From Buffer Type	To Buffer Type
O	3270	*FORMAT
O	3270	5250
O	*FORMAT	5250
I	5250	*FORMAT
I	5250	3270
I	*FORMAT	3270

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF5D53 E To and from buffers overlap.
CPF5D54 E Value &1 for operation parameter not valid.
CPF5D55 E Value &1 for to buffer parameter not valid.
CPF5D56 E Value &1 for from buffer type parameter not valid.
CPF5D57 E Combination of parameter values not valid.
CPF5D58 E Translation session handle parameter value not valid.
CPF5D59 E Value &1 for from buffer length parameter not valid.
CPF5D5A E Value &1 for the to buffer length parameter not valid.
CPF5D5C E 3270 data stream in from buffer not valid.

An error was found while translating the 3270 data stream in the from buffer. The error code for translation was &1.

X'0002' A 3270 command or order that is not supported or not valid was detected in the data stream.
X'0003' A parameter or address that is not valid was detected in the 3270 data stream.
X'0004' Excess fields were detected in the data stream. A certain number of these fields are allowed based on the device specified on the QD0STRTS call. This number of fields was exceeded.
X'0021' A set buffer address order is missing after a row-column AID code.
X'0863' A character set attribute that is not valid was found in the data stream.

CPF5D5D E 5250 data stream in from buffer not valid.

An error was found while translating the 5250 data stream in the from buffer. The error code for the translation was &1.

X'0001' A 5250 AID code that was not correct was found in the data stream.
X'0020' A cursor position that was not valid was detected in the 5250 data stream.
X'0021' A set buffer address order is missing after a row-column AID code.
X'0022' A set buffer address order that was not valid was found in the data stream.

CPF5D5E E Return code in formatted buffer indicates error. Codes returned in this message are listed in *SNA Upline Facility Programmer's Guide*.
CPF5D5F E Data integrity error in the from buffer value. The error code for the translation is &1. The possible error codes are:

X'0023' Character not valid.
X'0050' Shift out (X'0E') and shift in (X'0F') not correctly balanced in a DBCS session.
X'0051' Shift out (X'0E') and shift in (X'0F') in a DBCS field.
X'0052' The dead position in a DBCS field is not null.
X'0053' A DBCS character is not valid.

CPF5D60 E To buffer not large enough for translation output.
CPF5D62 E Error occurred in translation routines.
CPF5D63 E Data integrity error in formatted buffer. The error code for the translation is &1. The possible error codes are:

X'0023' Character not valid.
X'0050' Shift out (X'0E') and shift in (X'0F') not correctly balanced in a DBCS session.
X'0051' Shift out (X'0E') or shift in (X'0F') in a DBCS field.
X'0052' The dead position in a DBCS field is not null.
X'0053' A DBCS character is not valid.

CPF5D64 E To buffer length not valid for the to buffer value.
CPF5D65 E From buffer length not valid for the from buffer value.
CPF5D67 E Severe error occurred while addressing parameter list.

Part 3. Database File APIs

Chapter 9. Database File APIs

The database file APIs retrieve specific information about OS/400 files. The database file APIs include the following:

- **List Database File Members** (QUSLMBR) generates a list of database file members and places the list in a user space
- **List Database Relations** (QDBLDBR) provides information on how files and members are related to a specified database file
- **List Fields** (QUSLFLD) generates a list of fields within a specified file record format name
- **List Record Formats** (QUSLRCD) generates a list of record formats contained within a specified file
- **Query** (QQQQRY) gets a set of database records that satisfy a database query request
- **Retrieve File Description** (QDBRTVFD) provides complete and specific information about a file on a local or remote system
- **Retrieve Member Description** (QUSRMBRD) returns specific information about a single database file member

With the exception of QDBLDBR and QUSRMBRD, these APIs work with files that are either local or remote. Local files are files that are on the AS/400 system where the program is running. Remote files are files on a target (remote) system that are accessed using a distributed data management (DDM) file on a source (local) system. DDM files provide the information needed for a local system to locate a remote system and to access data in the remote system's database files. The QDBLDBR and QUSRMBRD APIs only work with local database files.

When you are calling these APIs from your high-level language (HLL) program, you must specify whether or not to use file override processing on your local or remote files. However, the QDBLDBR API does not support overrides.

The database file APIs use the standard user space format for the lists of information they return. If you are not yet familiar with this format, see "User Space Format for List APIs" on page 2-7 before using these APIs.

The following sections describe each API in detail.

List Database File Members (QUSLMBR) API

Parameters

Required Parameter Group:

1	User space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	Database file name and library	Input	Char(20)
4	Member name	Input	Char(10)
5	Override processing	Input	Char(1)

Optional Parameter:

6	Error code	I/O	Char(*)

The List Database File Members (QUSLMBR) API generates a list of database file members and places the list in a specified user space. When you specify a generic member name, you can generate a subset of the member list. You can use the QUSLMBR API with database file types *PF, *LF, and *DDMF. The generated list replaces any existing lists in the user space. The file members listed in the user space are not in any predictable order. To retrieve additional information about each member in the list, see "Retrieve Member Description (QUSRMBRD) API" on page 9-17.

You can use the QUSLMBR API to:

- List members more quickly than by using the *MBRLIST value on the TYPE parameter of the Display File Description (DSPFD) command.
- Ensure that the last date the source was changed matches the date of the source used to create the object.

Authorities and Locks

User Space Authority	*CHANGE
Library Authority	*USE
File Authority	*OBJOPR
User Space Lock	*EXCLRD
File Lock	*SHRRD

Required Parameter Group

User space and library name
INPUT; CHAR(20)
The user space that is to receive the created list. The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. You can use these special values for the library name:

***CURLIB** The job's current library

***LIBL** The library list

Format name
INPUT; CHAR(8)
The content and format of the information returned for each member. The possible format names are:

MBRL0100 Member name
MBRL0200 Member name and source information This format requires more processing than the MBRL0100 format.

For more information, see "MBRL0100 List Data Section" or "MBRL0200 List Data Section."

Database file name and library
INPUT; CHAR(20)
The name of the database file whose member names are to be placed in the list. The first 10 characters contain the database file name, and the second 10 characters contain the name of the library where the file is located.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Member name
INPUT; CHAR(10)
A specific member name, a generic member name, or this special value:

***ALL** All members

Override processing
INPUT; CHAR(1)
Whether overrides are to be processed. The following character values are used:

0 No override processing
1 Override processing

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated Lists

The file member list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section:
  - MBRL0100 format
  - MBRL0200 format

For details about the user area and generic header, see "User Space Format for List APIs" on page 2-7. For details about the remaining items, see the following sections. For detailed descriptions of the fields in the list returned, see "Field Descriptions" on page 9-3.

Input Parameter Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	File name specified
38	26	CHAR(10)	File library name specified
48	30	CHAR(10)	Member name specified
58	3A	CHAR(1)	Override processing

Header Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	File attribute
30	1E	CHAR(50)	File text description
80	50	BINARY(4)	Total number of members in file
84	54	CHAR(1)	Source file

MBRL0100 List Data Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Member name used

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Appendix A, "Examples."

MBRL0200 List Data Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Member name used
10	A	CHAR(10)	Source type
20	14	CHAR(13)	Creation date and time
33	21	CHAR(13)	Last source change date and time
46	2E	CHAR(50)	Member text description

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Appendix A, "Examples."

Field Descriptions

Creation date and time. The date and time the member was created. The format of this field is in the CYYMMDDHHMMSS format where:

C	Century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
YY	Year
MM	Month
DD	Day
HH	Hour
MM	Minute
SS	Second

File attribute. The type of file found:

PF	Physical file
LF	Logical file
DDMF	Distributed data management file

File library name specified. The name of the library containing the file whose member names are to be placed in the list.

File library name used. The name of the library containing the file whose member names are placed in the list.

File name specified. The name of the file specified in the call to the API.

File name used. The name of the file whose member names are placed in the list.

File text description. The description of the file.

Format name. The content and format of the information returned for each member. The possible values are:

MBRL0100	Member
MBRL0200	Member and source information

Last source change date and time. The date and time that this source member was last changed. This field is in the CYYMMDDHHMMSS format where the format is the same as for the creation date and time field.

Member name specified. The name of the member specified in the call to the API.

Member name used. The name of a member found in the file.

Member text description. The text description of the member.

Override processing. Whether overrides are to be processed. The possible values are:

0	No override processing
1	Override processing

Source file. Whether the file is a source file or a data file. The possible values are:

0	Data file
1	Source file

Source type. The type of source member if this is a source file. Some possible values are:

- CL
- COBOL
- RPG
- TXT

Total number of members in file. The total number of members in the file specified.

User space library name. The name of the library that contains the user space that is to receive the generated list.

User space name. The name of the user space that is to receive the generated list.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C20 E Error found by program &1.
  CPF0600 D All CPF06xx messages could be signaled. xx is from 01 to FF.
  CPF3200 D All CPF32xx messages could be signaled. xx is from 01 to FF.
  CPD3C21 D Format name &1 is not valid.
  CPD3C25 D Value &1 for file override parameter is not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C22 E Cannot get information about file &1.
CPF3C23 E Object &1 is not a database file.
CPF3C25 E Value &1 for file override parameter is not valid.
CPF3C27 E Cannot get information about member &3 from file &1.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.

List Database Relations (QDBLDBR) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format	Input	Char(8)
3	Qualified file name	Input	Char(20)
4	Member	Input	Char(10)
5	Record format	Input	Char(10)
6	Error code	I/O	Char(*)

The List Database Relations (QDBLDBR) API gives relational information about database files. The information identifies the physical and logical files that are dependent on a specific file, files that use a specific record format, or file members that are dependent on a specific file member. The information is placed in a user space specified by you.

Similar in function to the Display Database Relations (DSPDBR) command, this API allows more input parameter values than does the command. Also, your program can have more direct access to the information put in the user space by this API than when the command places similar information in an output file.

The information generated by this API replaces any existing information in the user space. It does not append information to any information already in the user space. If the space is bigger than needed, the contents of the remainder of the space are not changed. If the space is not big enough, it is extended.

Authorities and Locks

User Space Authority	*CHANGE
User Space Library Authority	*USE
User Space Lock	*EXCLRD
File Authority	*USE
File Library Authority	*USE
File Lock	*SHRNUPD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The user space that is to receive the database relations information. The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format
INPUT; CHAR(8)
The content and format of the information to be returned about the specified file, member, or record format. One of the following format names may be used:

DBRL0100 File information
DBRL0200 Member information
DBRL0300 Record format information

For more information, see "DBRL0100 Format (File)" on page 9-5, "DBRL0200 Format (Member)" on page 9-5, or "DBRL0300 Format (Record Format)" on page 9-5.

Qualified file name
INPUT; CHAR(20)
The name of the file for which database relations information is to be extracted. The first 10 characters contain the file name, and the second 10 characters contain the name of the library where the file is located. The file name cannot be a DDM file.

The file name can be a specific file name, a generic name, or the following special value:

***ALL** All files

You can use these special values for the library name:

***ALL** All libraries in the system
***ALLUSR** All nonsystem libraries
***CURLIB** The job's current library
***LIBL** The library list
***USRLIBL** Libraries listed in the user portion of the library list

Member
INPUT: CHAR(10)
The name of the member to be used for retrieving database relations for format DBRL0200. This value can be a specific member name, a generic member name, or one of the following special values:

***FIRST** Information about the first member (in the order created) in the specified file or files is to be provided.
***LAST** Information about the last member (in the order created) in the specified file or files is to be provided.
***ALL** Information about all members in the specified files is to be provided.

This parameter is ignored for formats DBRL0100 and DBRL0300.

Record format
INPUT; CHAR(10)
The name of the record format to be used for retrieving database relations for format DBRL0300. This value can be a specific record format, a generic record format, or the following special value:

***ALL** All record formats in the specified file

This input is ignored for formats DBRL0100 and DBRL0200.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated List

The database relations list consists of an input parameter section and one of three possible formats for the list data section. The three formats are determined by the kind of information you are looking for. The format names are:

DBRL0100 Database relations (file)
DBRL0200 Database relations (member)
DBRL0300 Database relations (record format)

The layout of the contents of the user space is determined by the format used. The following tables show how the contents of the input parameter section and the data format sections are organized. For descriptions of each field, see "Field Descriptions."

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	File name specified
38	26	CHAR(10)	File library name specified
48	30	CHAR(10)	Member name specified
58	3A	CHAR(10)	Record format name specified

DBRL0100 Format (File): The structure of the information returned is determined by the value specified for the format name. The DBRL0100 format includes information on files dependent on the file specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	Dependent file name
30	1E	CHAR(10)	Dependent library name
40	28	CHAR(1)	Dependency type
41	29	CHAR(3)	Reserved
44	2C	BINARY(4)	Join reference number

DBRL0200 Format (Member): The structure of the information returned is determined by the value specified for the format name. The DBRL0200 format includes information on files and members dependent on the file member specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	Member name used
30	1E	CHAR(10)	Dependent file name
40	28	CHAR(10)	Dependent library name
50	32	CHAR(10)	Dependent member name
60	3C	CHAR(1)	Dependency type
61	3D	CHAR(3)	Reserved
64	40	BINARY(4)	Join reference number
68	44	BINARY(4)	Join file number

DBRL0300 Format (Record Format): The structure of the information returned is determined by the value specified for the format name. The DBRL0300 format includes information on files dependent on the record format specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	Record format name used
30	1E	CHAR(10)	Dependent file name
40	28	CHAR(10)	Dependent library name

Field Descriptions

Dependency type. How a file or member is related to the file or member specified with the QDBLDBR API. Possible values are:

blank No dependent files or members were found for the specified file.
D The dependent file or member is dependent on the data in the specified file or member that was extracted.
I The dependent file member is sharing the access path of the file that the information was extracted from.
O If an access path is shared, one of the file members is considered the owner. The owner of the access path is charged with the storage used for the access path. If the member displayed is designated the owner, one or more

file members are designated with an I for access path sharing.

V — The SQL view or member is dependent on the specified SQL view.

Dependent file name. The name of the file that is dependent on the file specified using the QDBLDBR API. If no dependent files are found for the file specified, the dependent file name is *NONE.

Dependent library name. The name of the library that the dependent file is in. If there are no dependent files found for the file specified, the dependent library name is blank.

Dependent member name. The name of the file member that is dependent on the file member specified using the QDBLDBR API. If no dependent members are found for the member specified, the dependent member name is *NONE.

File library name specified. The name of the library containing the file for which the database relations information is requested.

File library name used. The name of the library containing the file used to extract the database relations information in this list entry.

File name specified. The name of the file for which the database relations information is to be extracted.

File name used. The name of the file used to extract the database relations information in this list entry.

Format name. The name of the format in which the database relations information is returned to the user space.

Join file number. If the file for which database relations information is being extracted is a join logical file, this is the ordinal number of the file in the JFILE to which the dependency relates. The join file number is zero if either of the following are correct:

- No dependent files are found for the file specified.
- The file for which the information is being extracted is not a join file.

Join reference number. If the dependent file listed is a join logical file, this is the ordinal number of the file in the JFILE to which this dependency relates. The join reference number is zero if either of the following are correct:

- No dependent files are found for the file specified.
- The dependent file is not a join file.

Member name specified. The name of the member for which the information is extracted.

Member name used. The name of the member used to extract the database relations information in this list entry.

Record format name specified. The name of the record format for which the information is displayed.

Record format name used. The name of the record format used to extract the database relations information in this list entry.

Reserved. An ignored field.

User space library name. The name of the library that contains the user space that receives the database relations information requested.

User space name. The name of the user space that receives the database relations information requested.

Error Messages

CPF2207	E	Not authorized to use object &1 in library &3 type &2.
CPF3227	E	Record format &3 for file &1 not found.
CPF326C	E	Value &1 for file name parameter is not a valid special value.
CPF326D	E	Value &1 for member name parameter is not a valid special value.
CPF326E	E	Value &1 for record format name parameter is not a valid special value.
CPF3C20	E	Error found by program &1.
CPF3C21	E	Format name &1 is not valid.
CPF3C23	E	Object &1 is not a database file.
CPF3CF1		Error code parameter not valid.
CPF5715	E	File &1 in library &2 not found.
CPF5815	E	Member &3 for file &1 in library &2 not found.
CPF9801		Object &1 in library &2 not found.
CPF9802		Not authorized to object &2 in &3.
CPF9803		Cannot allocate object &1 in library &3.
CPF9807		One or more libraries in library list deleted.
CPF9808		Cannot allocate one or more libraries in library list.
CPF9810		Library &1 not found.
CPF9811		Program &1 in library &21 not found.
CPF9812		File &1 in library &21 not found.
CPF9820		Not authorized to use library &1.
CPF9821		Not authorized to use program &1 in library &2.
CPF9822		Not authorized to file &1 in library &2.
CPF9830		Cannot assign library &1.

List Fields (QUSLFLD) API

Parameters

Required Parameter Group:

1	User space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	File and library name	Input	Char(20)
4	Record format name	Input	Char(10)
5	Override processing	Input	Char(1)

Optional Parameter:

6	Error code	I/O	Char(*)

The List Fields (QUSLFLD) API generates a list of fields within a specified file record format name. The list of fields is placed in a specified user space. The generated

list replaces any existing list in the user space. You can use the QUSLFLD API only with database file types, such as *PF, *LF, and *DDMF, and device file types, such as *ICFF and *PRTF.

You can use the QUSLFLD API to:

- Generate a list of field format names.
- Gather additional information about specific field formats.
- Create a product similar to the Structured Query Language (SQL) using the Open Query File (OPNQRYF) command.
- Create applications similar to the data file utility (DFU).
- Create a compiler supporting externally described data.
- Create applications that use data defined to the system.

Authorities and Locks

User Space Authority	*CHANGE
Library Authority	*USE
File Authority	*OBJOPR
User Space Lock	*EXCLRD
File Lock	*SHRRD

Required Parameter Group

User space and library name
INPUT; CHAR(20)
The name of the user space that is to receive the created list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The format of the information returned. You must use the following format name:

FLDL0100 Field information

For more information, see "Format of the Generated List."

File and library name
INPUT; CHAR(20)
The file whose member names are to be placed in the list, and the library in which it is located. The first 10 characters contain the file name, and the second 10 characters contain the library name.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Record format name
INPUT; CHAR(10)
The record format name whose fields are to be returned.

Override processing
INPUT; CHAR(1)
Whether overrides are to be processed. The possible values are:

0 No override processing
1 Override processing

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated List

The field list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- The FLDL0100 list data section

For details about the user area and generic header, see "User Space Format for List APIs" on page 2-7. For details about the remaining items, see the following sections. For descriptions of each field in the list returned, see "Field Descriptions" on page 9-8.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	File name specified
38	26	CHAR(10)	File library name specified
48	30	CHAR(10)	Record format name specified
58	3A	CHAR(1)	Override processing

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used

Offset			
Dec	Hex	Type	Field
20	14	CHAR(10)	File type
30	1E	CHAR(10)	Record format name used
40	28	BINARY(4)	Record length
44	2C	CHAR(13)	Record format ID
57	39	CHAR(50)	Record text description

FLDL0100 List Data Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Field name
10	A	CHAR(1)	Data type
11	B	CHAR(1)	Use
12	C	BINARY(4)	Output buffer position
16	10	BINARY(4)	Input buffer position
20	14	BINARY(4)	Field length in bytes
24	18	BINARY(4)	Digits
28	1C	BINARY(4)	Decimal position
32	20	CHAR(50)	Field text description
82	52	CHAR(2)	Edit code
84	54	BINARY(4)	Edit word length
88	58	CHAR(64)	Edit word
152	98	CHAR(20)	Column heading 1
172	AC	CHAR(20)	Column heading 2
192	C0	CHAR(20)	Column heading 3

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Appendix A, "Examples."

Field Descriptions

Column heading 1. The description of the first column heading for this field. It contains blanks if the heading is not defined.

Column heading 2. The description of the second column heading for this field. It contains blanks if the heading is not defined.

Column heading 3. The description of the third column heading for this field. It contains blanks if the heading is not defined.

Data type. The type of field:

A Alphanumeric (character)
P Packed decimal
S Zoned decimal
B Binary
F Floating point
X Alphabetic only (character)
Y Numeric only
N Numeric shift
I Inhibit entry
W Katakana
D Digits only
M Numeric only
J Double-byte character set (DBCS) data only
E Either DBCS or alphanumeric
O (Open) Both DBCS and alphanumeric

Decimal position. The number of decimal positions. This entry is zero if the field is not numeric.

Digits. The number of numeric digits. This entry is zero if the field is not numeric.

Edit code. The field edit code.

Edit word. The field edit word.

Edit word length. The length of the edit word used.

Field length in bytes. The number of bytes the field occupies.

Field name. The name of the field the entry describes.

Field text description. The description of the field.

File library name specified. The library specified in the call to the API.

File library name used. The name of the library that contained the file.

File name specified. The file specified in the call to the API.

File name used. The name of the file where the member list was found.

File type. The type of file found.

BSCF Binary synchronous communications (BSC) file
CMNF Communications file
DDMF Distributed data management file
DKTF Diskette file
DSPF Display file
ICFF Intersystems communications function file
LF Logical file
MXDF Mixed file
PF Physical file
PRTF Printer file
SAVF Save file
TAPF Tape file

Format name. The content and format of the information returned for each field. The only possible value is:

FLDL0100 Field information

Input buffer position. The field's position within the input record.

Output buffer position. The field's position within the output record.

Override processing. Whether overrides are to be processed. The possible values are:

0 No override processing
1 Override processing

Record format ID. The record format identifier.

Record format name specified. The record format specified in the call to the API.

Record format name used. The name of this record format.

Record length. The length of this record format.

Record text description. The text description of this record format.

Use. How the field is used:

I Input
O Output
B Both input and output
N Neither

User space library name. The name of the library that contains the user space that is to receive the generated list.

User space name. The name of the user space that is to receive the generated list.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C20 E Error found by program &1.
CPF0600 D All CPF06xx messages could be signaled. xx is from 01 to FF.
CPF3200 D All CPF32xx messages could be signaled. xx is from 01 to FF.
CPF3C21 D Format name &1 is not valid.
CPD3C21 D Format name &1 is not valid.
CPD3C25 D Value &1 for file override parameter is not valid.
CPF3C22 E Cannot get information about file &1.
CPF3C25 E Value &1 for file override parameter is not valid.
CPF3C28 E Record format &3 in file &1 not found.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.

List Record Formats (QUSLRCD) API

Parameters

Required Parameter Group:

1	User space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	File and library name	Input	Char(20)
4	Override processing	Input	Char(1)

Optional Parameter:

5	Error code	I/O	Char(*)

The List Record Formats (QUSLRCD) API generates a list of record format information contained within the specified file and places the list in a specified user space. The created list replaces any existing lists in the user space.

You can use the QUSLRCD API with database file types, such as *PF, *LF, and *DDMF, and device file types, such as *DSPF, *TAPF, *DKTF, *PRTF, *SAVF, and *ICFF.

Authorities and Locks

User Space Authority	*CHANGE
Library Authority	*USE
File Authority	*OBJOPR
User Space Lock	*EXCLRD
File Lock	*SHRRD

Required Parameter Group

User space and library name
INPUT; CHAR(20)
The name of the user space that is to receive the generated list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The format of the information returned. The possible format names are:

RCDL0100 Record format name only.
RCDL0200 Record format name and additional information. This format requires more system paging and takes longer to produce than the RCDL0100 format.

For more information, see "RCDL0100 List Data Section" on page 9-10 or "RCDL0200 List Data Section" on page 9-10.

File and library name
INPUT; CHAR(20)
The name of the file whose record format names are placed in the list, and the library in which it is located. The first 10 characters contain the file name, and the

second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Override processing
INPUT; CHAR(1)
Whether overrides are to be processed. The possible values are:

0 No override processing
1 Override processing

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated List

The record format list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

For details about the user area and generic header, see "User Space Format for List APIs" on page 2-7. For details about the other items, see the following sections. For descriptions of each field, see "Field Descriptions."

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	File name specified
38	26	CHAR(10)	File library name specified
48	30	CHAR(1)	Override processing

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	File name used
10	A	CHAR(10)	File library name used
20	14	CHAR(10)	File type
30	1E	CHAR(50)	File text description

RCDL0100 List Data Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Record format name

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Appendix A, "Examples."

RCDL0200 List Data Section

Offset		Type	Description
Dec	Hex		
0	0	CHAR(10)	Record format name
10	A	CHAR(13)	Record format ID
23	17	CHAR(1)	Reserved
24	18	BINARY(4)	Record length
28	1C	BINARY(4)	Number of fields
32	20	CHAR(50)	Record text description

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Appendix A, "Examples."

Field Descriptions

File library name specified. The name of the file library specified in the call to the API.

File library name used. The name of the library that contained the file. If the library requested was *LIBL or *CURLIB, this field contains the name of the library where the system found the file.

File name specified. The name of the file specified in the call to the API.

File name used. The name of the file whose record formats are listed. If override processing was requested, this is the actual file.

File text description. The text description of the file.

File type. The type of file found:

BSCF Binary synchronous communications (BSC) file
CMNF Communications file
DSPF Display file
DDMF Distributed data management file
DKTF Diskette file
ICFF Intersystems communications function file
LF Logical file
MXDF Mixed file

PF	Physical file
PRTF	Printer file
SAVF	Save file
TAPF	Tape file

Format name. The name of the format used to list records. The possible values are:

RCDL0100	Record format name only
RCDL0200	Record format name and additional information

Number of fields. The number of fields contained in this record format. You can use the List Field Description (QUSLFLD) API to retrieve field information about this record.

Override processing. Whether overrides are to be processed. The possible values are:

0	No override processing
1	Override processing

Record format ID. The record format identifier.

Record format name. The name of this record format.

Record length. The length of this record format.

Record text description. The text description of this record format.

Reserved. An ignored field.

User space library name. The name of the library that contains the user space that is to receive the generated list.

User space name. The name of the user space that is to receive the generated list.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C20 E Error found by program &1.
CPD3C21 D Not authorized to program &1 in library &2.
CPD3C25 D Value &1 for file override parameter is not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C22 E Cannot get information about file &1.
CPF3C25 E Value &1 for file override parameter is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.

Query (QQQQRY) API

Parameters

Required Parameter Group:

1	Query option requested	Input	Char(10)
2	User file control block	I/O	Char(*)
3	Query definition template	I/O	Char(*)
4	Literal values	I/O	Char(*)
5	Access plan control block	I/O	Char(48)
6	Error code	I/O	Char(*)

The Query (QQQQRY) API gets a set of database records that satisfies a database query request. Using this API you can do all the things you could do with the Open Query File (OPNQRYF) command. You can also perform subqueries, perform unions, and use SQL host variables.

The QQQQRY API can be used to do any combination of the following database functions:

- Join records from more than one file, member, and record format. The join operation that is performed may be equal or nonequal in nature.
- Calculate new field values by using numeric and character operations on field values and constants.
- Group records by like values of one or more fields, and calculate aggregate functions, such as minimum field value and average field value, for each group.
- Select a subset of the available records. Selection can be done both before and after grouping the records.
- Arrange result records by the value of one or more key fields.

You can use this API to run a query, create an access plan, or get information from the query definition template. When you run the query, the API uses the information you provide with the query definition template to extract information and data from the database. Creating an access plan makes it possible to run the future queries with better performance. Checking the query definition template allows you to validate this query definition template.

The record format definition is part of the query definition template and can be created and saved with extracted information by the Retrieve File Description (QDBRTVFD) API. Another part of the query definition template is the access plan for the query.

You can use this API with the RUNQRY value of the query option requested parameter to build an access plan to more efficiently run a query more than once. You can then use the access control block parameter to point to the access plan. This greatly improves the time it takes to run subsequent queries using this API and the RUNQRY option. Every time a query is run, the system first checks to see if an access plan has been specified. If one has, that is what is used to get the data requested by the query. If no access plan has been specified, a new one is built dynamically.

Authorities and Locks

User Space Authority *CHANGE

Library Authority	*USE
File Authority	*OBJOPR
User Space Lock	*SHRRD

Required Parameter Group

Query option requested
INPUT; CHAR(10)
One of three options to be used:

RUNQRY	Run query
CRTQAP	Create access plan
CHKQDT	Check query definition templates

User file control block
I/O; CHAR(*)
One or more selected options for input and output of the specified query. This parameter is only required when using the RUNQRY query option. See "User File Control Block (QDBUFCB) Structure" on page 9-13 for a list of available options.

Query definition template
I/O; CHAR(*)
The information required to create objects that are used to query a database. It contains feedback information from the creation of objects. If a pointer to the access plan is specified, the corresponding query definition templates must also be specified.

Literal values
I/O; CHAR(*)
This parameter is used to put into effect SQL host variables. When SQL host variables are used, this is a list of constant values used to run a query. This parameter is ignored if a pointer is not specified. Once the literal value is specified it must always be specified.

Access plan control block
I/O; CHAR(48)
A string of bytes that point to the access plan control block and give the size the access plan requires. This parameter must be specified for the RUNQRY query option when you want to specify an access plan and the CRTQAP query option. The format for this parameter is:

PTR(SPP)	A space pointer that indicates the area of storage that contains the access plan. This area must begin on a 16-byte boundary and be all zeros.
Bin(4)	The size of storage needed to contain the access plan.
Char(28)	Reserved.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Data Structures

The QQQQRY API uses information in four structures for performing a query. All structures are used together to perform the function you have selected using the query option requested parameter. The names of these structures are:

QDBQDT	Query definition template
QDBFMTD	Format definition
QDBUFCB	User file control block
QQQVALS	Values for query variable fields

The following sections show you in a general way how this information is structured.

Query Definition Template (QDBQDT): This template provides information about the query that is to be performed. Figure 9-1 shows the general layout of this format. To get a description of all the fields contained in this structure and to determine the offsets, see the include source supplied on the AS/400 system. You can see this source in source file QATTSYSC, member name TOPQQAPI, in the QUSRTOOL library.

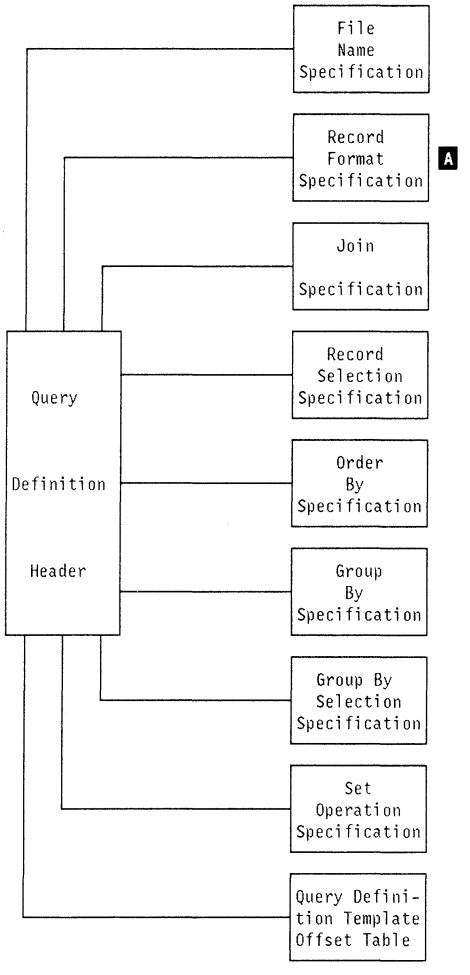


Figure 9-1. QDBQDT Format

Notice the box marked with an **A** in Figure 9-1. The topic "Format Definition (QDBFMTD)" on page 9-13 provides the layout of the entire record format specification.

Format Definition (QDBFMTD): The record format definition (QDBfmtd) for the QQQQRY API is the same structure used by the Retrieve File Description (QDBRTVFD) API called FILD0200. Figure 9-2 provides an overview of this structure. To get a description of all the fields contained in this structure and to determine the offsets, see the include source supplied on the AS/400 system. You can see this source in source file QATTSYSC, member name TOPDBAPI, in the QUSRTOOL library.

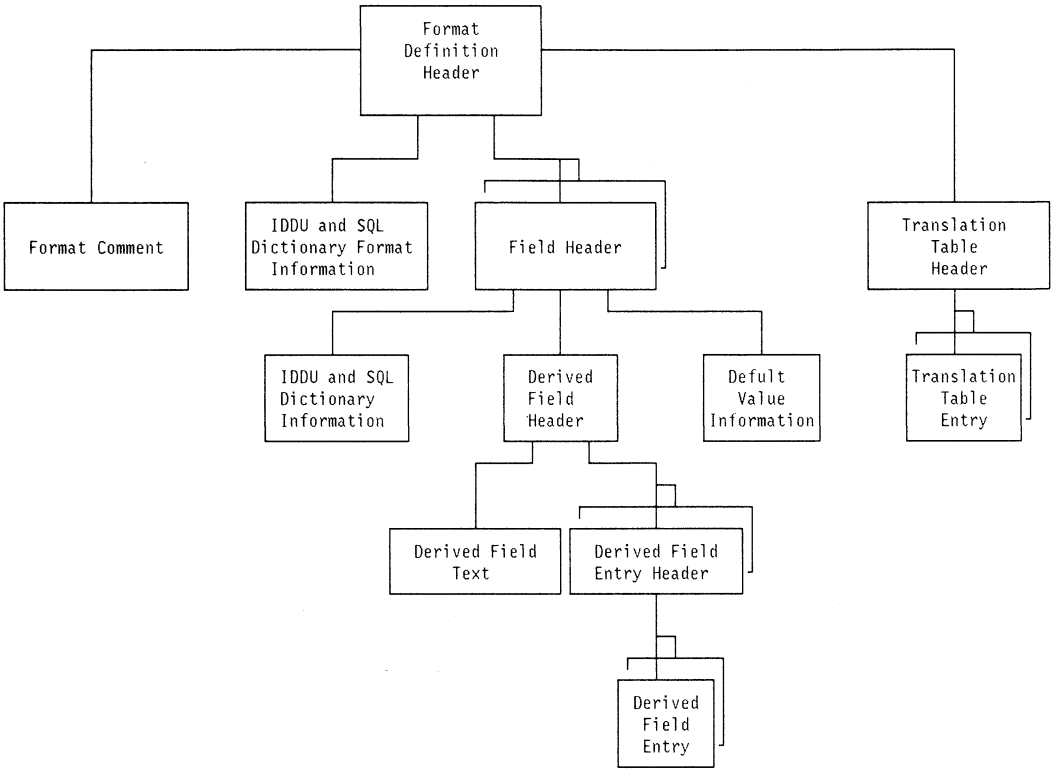


Figure 9-2. QDBFMTD Format

User File Control Block (QDBUFCB) Structure: This structure holds information from the user function control block (UFCB). It contains selected options for the input and output of the specified query. The options available include:

- Sequence only
- Commitment control
- Block records
- Keyed feedback
- Record length
- Open options
- Release number
- Version number
- Invocation mark count
- AS/400 system environment
- Null-capable fields
- File dependency
- Level check
- Record format specifications
- Secure
- Shared

In addition, some validity checking is done for this UFCB. CPF4297 is issued if any reserved space in the header of the QDBUFCB format is not zero.

To get a description of all the fields contained in this structure and to determine the offsets, see the include source supplied on the AS/400 system. You can see this source in source file QATTSYSC, member name TOPQQAPI, in the QUSRTOOL library.

Value For Query Variable Fields (QQQVALS) Structure: The structure is used to supply the values for the variable fields used by the QQQQRY API. To get a description of all the fields contained in this structure and to determine the offsets, see the include source supplied on the AS/400 system. You can see this source in source file QATTSYSC, member name TOPQQAPI, in the QUSRTOOL library.

Error Messages

CPF3C20 E Error found by program SCRIPT
CPF3C21 E Format name SCRIPT is not valid.

CPF3C22 E	Cannot get information about file SCRIPT
CPF3C23 E	Object SCRIPT is not a database file.
CPF3C25 E	Value SCRIPT for file override parameter is not valid.
CPF3C27 E	Cannot get information about member &3 from file SCRIPT
CPF3C36 E	Number of parameters, &1, entered for this API was not valid.
CPF3CF1 E	Error code parameter not valid.
CPF4297	Value specified in UFCB (User File Control Block) is not valid at offset &1.

Retrieve File Description (QDBRTVFD) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Returned file and library name	Output	Char(20)
4	Format name	Input	Char(8)
5	File and library name	Input	Char(20)
6	Record format name	Input	Char(10)
7	Override processing	Input	Char(1)
8	System	Input	Char(10)
9	Format type	Input	Char(10)
10	Error code	I/O	Char(*)

The Retrieve File Description (QDBRTVFD) API allows you to get complete and specific information about a file on a local or remote system. The information is returned to a receiver variable in either a file definition template or a format definition mapping. The file definition template provides more complete information about a database file than the Display File Description (DSPFD) command. The format definition provides complete information on the record formats of the file.

The record format definition is used with the Query (QQQQRY) API to get data from a file. You can run the QDBRTVFD API to build a record format definition that is later used to run a query. This record format definition can be used several times to extract information from a database, making the Query API run faster. If the record format definition is not created prior to running a query, the QQQQRY API must create one when it runs.

The extracted file information replaces any existing lists in main storage. It is not appended to any existing data. If the returned data does not fill the receiver variable, the contents of the remainder of the variable are not changed.

Authorities and Locks

Library Authority	*USE
File Authority	*OBJOPR
File Lock	*SHRNUP

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The receiver variable that is to receive the information requested. You can specify the size of the area smaller than the format requested as long as you specify the length of receiver variable parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results are not predictable. The minimum length is 8 bytes.

Returned file and library name
OUTPUT; CHAR(20)
The actual file and library name from which the file description has been extracted. If an override is active this file and library name may be different from the one entered with the API.

Format name
INPUT; CHAR(8)
The content and format of the information to be returned about the specified file, member, or format. You can use either of the following format names:

FILD0100 File description template
FILD0200 Format definition

See "Format of Generated Information" for a description of these formats.

File and library name
INPUT; CHAR(20)
The name of the file about which the information is to be extracted and the library in which it is located. The first 10 characters contain the file name, and the second 10 characters contain the library name.

You can use the following special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Record format name
INPUT; CHAR(10)
The name of the record format in the specified file that is to be used to generate the file description.

Override processing
INPUT; CHAR(1)
Whether overrides are to be processed. The following values are used:

0 No override processing
1 Override processing

System
INPUT; CHAR(10)
Whether the information that is returned is about a file on either a local or remote system, or both. The possible values are:

***LCL** The information returned is about local files only.
***RMT** The information returned is about remote files only.
***ALL** The information returned is about files on both the local and remote systems. For DDM files, the information returned is about the remote file that was named on the RMTFILE parameter of the Create DDM File (CRTDDMF) command.

Format type
INPUT; CHAR(10)
Whether the formats returned are internal or external:

***INT** The formats returned are internal.
***EXT** The formats returned are external.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of Generated Information

The QDBRTVFD API can be used to provide information in two formats:

FILD0100 File definition table
FILD0200 Format definition

The following sections provide an overview of each of these formats.

FILD0100 Format (Format Definition Template (FDT) header): FILD0100 provides detailed information about how the file is built. Figure 9-3 on page 9-16 shows how this information is organized. When more than one entry can appear, the figure indicates this as in **A**. To get a description of all the fields contained in this structure, and to determine the offsets, see the include source supplied on the AS/400 system. You can see this source in source file QATTSYSC, member name TOPDBAPI, in the QUSRTOOL library.

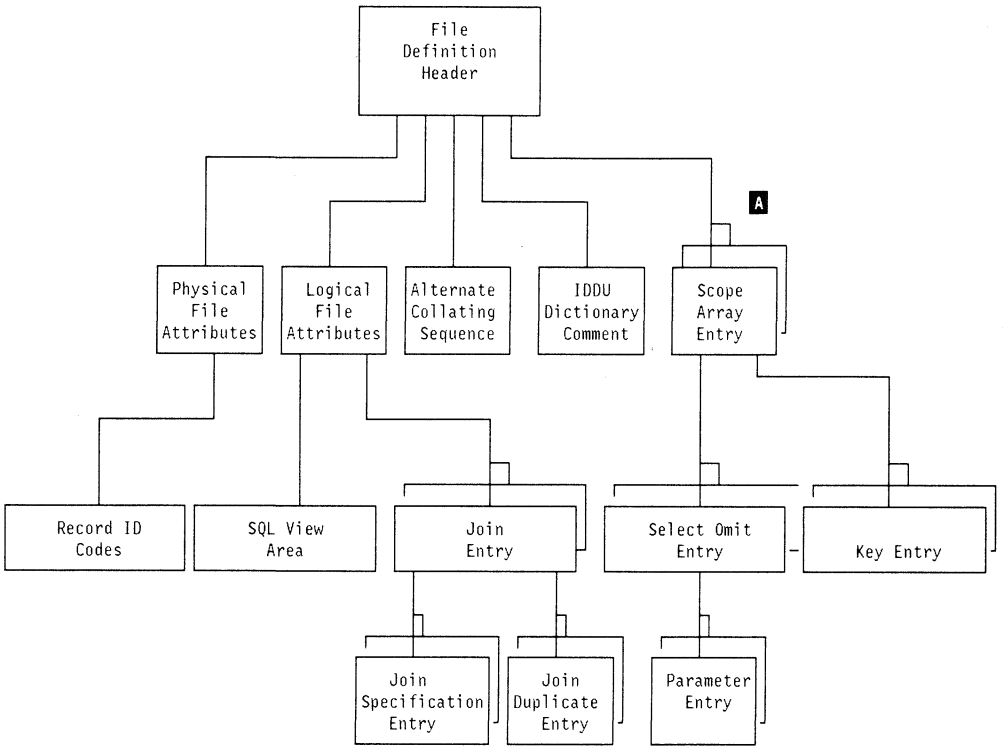


Figure 9-3. FILD0100 Format

FILD0200 Format (QDBFMTD Structure): FILD0200 provides the format used by the records of the specified file. This structure is used by the QQQQRY API to get data from the named file. Figure 9-4 on page 9-17 shows how this information is organized. When more than one entry can appear, the figure indicates this as in **A**. To get a description of all the fields contained in this structure and to determine the offsets, see the include source supplied on the AS/400 system. You can see this source in source file QATTSYSC, member name TOPDBAPI, in the QUSRTOOL library.

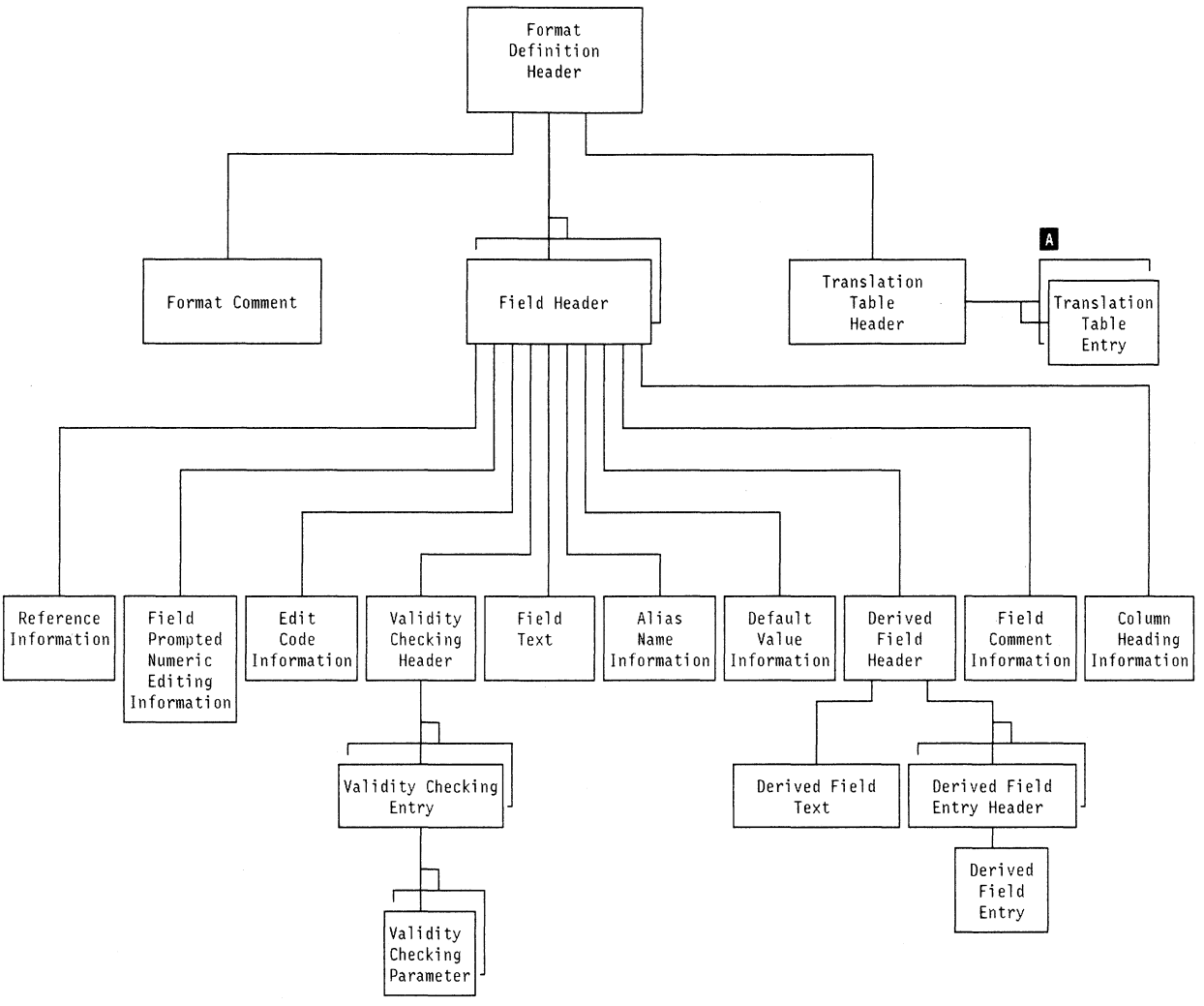


Figure 9-4. FILD0200 Format

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3021 E File &1 not allowed with SYSTEM(*RMT).
CPF3025 E File &1 not allowed with SYSTEM(*LCL).
CPF326F E Value &1 for system parameter is not valid.
CPF327A E Value &1 for format type parameter is not valid.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 is not valid.
CPF3C22 E Cannot get information about file &1.
CPF3C23 E Object &1 is not a database file.
CPF3C24 E Length of the receiver variable is not valid.
CPF3C25 E Value &1 for file override parameter is not valid.
CPF3CF1 E Error code parameter not valid.

Retrieve Member Description (QUSRMBRD) API

Retrieve Member Description (QUSRMBRD)

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Database file and library name	Input	Char(20)
5	Database member name	Input	Char(10)
6	Override processing	Input	Char(1)

Optional Parameter:

7	Error code	I/O	Char(*)

The Retrieve Member Description (QUSRMBRD) API retrieves specific information about a single database file member and returns the information to the calling program in a receiver variable. The length of the receiver variable determines the amount of data returned. You can only use the QUSRMBRD API with database file types *PF, *LF, and *DDMF.

You can use the QUSRMBRD API to:

- Retrieve specific information about a database file member that is specified to a calling program.
- Handle reorganization automatically when the deleted record space reaches the maximum specified.
- Ensure that the last date the source was changed matches the date the source was used to create the object.

Authorities and Locks

Library Authority	*USE
File Authority	*OBJOPR
File Lock	*SHRRD

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The receiver variable that is to receive the information requested. You can specify that the size of the area be smaller than the format requested as long as you specify the length of the receiver variable parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results are not predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The content and format of the information to be returned for each specified member. The following format names are valid:

MBRD0100 Member name and basic source information. This is similar to the information provided by the List Database File Members (QUSLMBR) API using format MBRL0200.
MBRD0200 Member name and expanded information. The additional information requires more system processing and takes longer to produce than the MBRD0100 format.
MBRD0300 Member name and full information. The additional information requires more system processing and takes longer to produce than the MBRD0200 format.

For more information, see "MBRD0100 Format," "MBRD0200 Format" on page 9-19 or "MBRD0300 Format" on page 9-19.

Database file and library name
INPUT; CHAR(20)
The name of the database file containing the specified member whose information is to be retrieved, and the library in which it is located. The first 10 characters contain the database file name, and the second 10 characters contain the library name.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Database member name
INPUT; CHAR(10)
The name of the database member for which information is to be retrieved. You can use the special values *FIRST and *LAST.

Override processing
INPUT; CHAR(1)
Whether overrides are to be processed. The possible values are:

0 No override processing
1 Override processing

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated Information

The file member description can be provided in one of three formats:

- MBRD0100
- MBRD0200
- MBRD0300

The structure of the information returned is determined by the value specified for the format name. For details about these formats, see the following sections. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 9-20.

MBRD0100 Format: The MBRD0100 format includes the file member list and source information shown in the following table.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Database file name
18	12	CHAR(10)	Database file library name
28	1C	CHAR(10)	Member name
38	26	CHAR(10)	File attribute
48	30	CHAR(10)	Source type
58	3A	CHAR(13)	Creation date and time
71	47	CHAR(13)	Last source change date and time

Offset		Type	Field
Dec	Hex		
84	54	CHAR(50)	Member text description
134	86	CHAR(1)	Source file

MBRD0200 Format: The MBRD0200 format includes the file member name and the expanded information shown in the following table.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the MBRD0100 format
135	87	CHAR(1)	Remote file
136	88	CHAR(1)	Logical file or physical file
137	89	CHAR(1)	ODP sharing
138	8A	CHAR(2)	Reserved
140	8C	BINARY(4)	Current number of records for all based-on members
144	90	BINARY(4)	Number of deleted records
148	94	BINARY(4)	Data space size
152	98	BINARY(4)	Access path size
156	9C	BINARY(4)	Number of based-on physical file members
160	A0	CHAR(13)	Change date and time
173	AD	CHAR(13)	Save date and time
186	BA	CHAR(13)	Restore date and time
199	C7	CHAR(7)	Expiration date
206	CE	CHAR(6)	Reserved
212	D4	BINARY(4)	Number of days used
216	D8	CHAR(7)	Date last used
223	DF	CHAR(7)	Use reset date
230	E6	CHAR(2)	Reserved
232	E8	BINARY(4)	Data space size multiplier
236	EC	BINARY(4)	Access path size multiplier
240	F0	CHAR(26)	Reserved

MBRD0300 Format: The MBRD0300 format includes the file member list and the full information shown in the following table. This includes some key fields that are applicable only to the file (not member) one might use, and fields unique to only the member.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the MBRD0200 format
266	10A	CHAR(1)	Join member
267	10B	CHAR(1)	Access path maintenance
268	10C	CHAR(10)	SQL file type
278	116	CHAR(1)	Reserved
279	117	CHAR(1)	Allow read operation
280	118	CHAR(1)	Allow write operation
281	119	CHAR(1)	Allow update operation
282	11A	CHAR(1)	Allow delete operation
283	11B	CHAR(1)	Reserved
284	11C	BINARY(4)	Records to force a write
288	120	BINARY(4)	Maximum percent deleted records allowed
292	124	BINARY(4)	Initial number of records
296	128	BINARY(4)	Increment number of records
300	12C	BINARY(4)	Maximum number of increments
304	130	BINARY(4)	Current number of increments
308	134	BINARY(4)	Record capacity
312	138	CHAR(10)	Record format selector program name
322	142	CHAR(10)	Record format selector library name
332	14C	CHAR(52)	Reserved
384	180	Array of CHAR(112)	Record format and based-on file list

Record Format and Based-On File List Entry: The last entry in the MBRD0300 format is the record format and based-on file list. There can be several entries with the information presented in the order as listed below. Since there can be several, it is not possible to list the exact offsets for the 112 bytes needed for each entry.

Offset		Type	Field
Dec	Hex		
		CHAR(10)	Based-on physical file name
		CHAR(10)	Based-on physical file library name
		CHAR(10)	Based-on physical file member name
		CHAR(10)	Format name
		BINARY(4)	Logical file record format number
		BINARY(4)	Current number of records
		BINARY(4)	Number of deleted records
		BINARY(4)	Access path size
		BINARY(4)	Access path size multiplier
		CHAR(1)	Access path shared
		CHAR(1)	Access path valid
		CHAR(1)	Access path held

Offset		Type	Field
Dec	Hex		
		CHAR(10)	Access path owner file name
		CHAR(10)	Access path owner library name
		CHAR(10)	Access path owner member name
		CHAR(19)	Reserved

Field Descriptions

Access path held. Indicates if rebuild of access path is held. More information can be found in the *CL Reference* manual under the Edit Rebuild Access Path (EDTRBDAP) command. Possible values are:

blank Not applicable unless the access path is for a join logical file or keyed file. Only indexes that are not valid can be held.
0 Access path is not held.
1 Access path is held.

Access path maintenance. Specifies, for files with key fields or join logical files, the type of access path maintenance used for all members of the physical or logical file.

The possible values are:

blank Does not apply unless the access path is for a join logical file or a keyed file.
0 The access path is updated each time a record is changed, added, or deleted from a member. Files that require unique keys are 0.
1 The access path is updated when the member is opened with records that have been added, deleted, or changed from the member since the last time the member was opened.
2 The access path is completely rebuilt each time a file member is opened. The access path is maintained until the member is closed, then the access path is deleted.

Access path owner file name. The file name that owns the access path. This field only applies to join logical files or keyed files.

Access path owner library name. The library in which the file resides that owns the access path. This field only applies to join logical files or keyed files.

Access path owner member name. The member within the qualified file name that owns the access path. This field only applies to join logical files or keyed files.

Access path shared. Whether an access path is shared. The possible values are:

blank Does not apply unless the access path is for a join logical file or keyed file.
0 Access path is not shared by other files.
1 Access path is shared by other files.

Access path size. The access path size in bytes for this file member. If the file member is not keyed, the value 0 is returned. DDM files, which are not from a System/38 or AS/400 system, return value 0.

Access path size multiplier. The value to multiply the access path size by to get its true size.

Access path valid. Whether the access path is valid. The possible values are:

blank Does not apply unless the access path is for a join logical file or a keyed file.
0 Index is valid.
1 Index is not valid and must be rebuilt.

Allow delete operation. Whether records in this file can be deleted. The possible values are:

Y Records in this file can be deleted.
N Records in this file cannot be deleted.

Allow read operation. Whether records in the physical file can be read. The possible values are:

Y Records in this file can be read.
N Records in this file cannot be read.

Allow update operation. Whether records in this file can be updated. The possible values are:

Y Records in this file can be updated.
N Records in this file cannot be updated.

Allow write operation. Whether records can be written to the file. The possible values are:

Y Records can be written to this file.
N Records cannot be written to this file.

Based-on physical file library name. The library in which the based-on physical file resides. This field is blank for a physical file.

Based-on physical file member name. The physical file member this logical file member is based on. The number of elements in this array is defined by the number of based-on physical file members field. This field is blank for a physical file.

Based-on physical file name. The name of the physical file that contains the data associated with the logical file member. This field is blank for a physical file.

Bytes available. The total format data length.

Bytes returned. The length of the data returned.

Change date and time. The date and time this member was changed. This field is in the CYYMMDDHHMMSS format, where:

C Century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
YY Year
MM Month
DD Day

| *HH* Hour
| *MM* Minute
| *SS* Second

| **Creation date and time**. The date and time the member was created. This field is in the CYYMMDDHHMMSS format, which is described in the change date and time field description.

| **Current number of increments**. The number of increments that have been added to the member size (data space size). This field is 0 for logical files because the number of increments only applies to physical files.

| **Current number of records**. The number of records that currently exist in this member. A keyed logical file member returns the number of index entries. A nonkeyed logical file member returns the number of records in the based-on physical file member.

| **Current number of records for all based-on members**. The number of records that currently exist in this member. A logical member returns the summarization of index entries.

| **Database file library name**. The name of the library that contains the file.

| **Database file name**. The name of the file from which the member list was retrieved.

| **Data space size**. The size of the space that contains the data of the file member, in bytes. A logical file returns a 0.

| **Data space size multiplier**. The value to multiply the data space size by to get its true size. Typically this is 1, but for large files, the value may be greater than 1. If the data space size multiplier is greater than 1, then the value in the data space size field is not the actual size of the file.

| **Date last used**. The century and date this member was last used. The date last used field is in the CYYMMDD format, where:

| *blank* *NONE
| *C* Century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
| *YY* Year
| *MM* Month
| *DD* Day

| **Expiration date**. The date that this member expires. This is in the CYYMMDD format, which is the same format described for the date last used field description.

| **File attribute**. The type of file found:

| *PF* Physical file
| *LF* Logical file
| *DDMF* Distributed data management file

| **Format name**. The definition of how data is structured in the records contained in a file. If this is a join logical file or SQL view file, the format name is only valid for the entry in the record format and based-on file member list array.

| **Increment number of records**. The maximum number of records that are automatically added to the member when the number of records in the member is greater than the initial member size. This field applies only to physical files and is 0 for logical files.

| **Initial number of records**. The number of records that can be written to each member of the file before the member size is automatically extended. This field applies only to physical files and is 0 for logical files.

| **Join member**. Whether the member's logical file member combines (in one record format) fields from two or more physical file members.

| *0* Not a join member
| *1* Join member

| **Last source change date and time**. The date and time that this source member was last changed. The last source changed date and time is in the CYYMMDDHHMMSS format, which is in the same format as for the change date and time field.

| **Logical file or physical file**. Whether the file is a logical or physical file. The possible values are:

| *0* Member retrieved from a physical file
| *1* Member retrieved from a logical file

| **Logical file record format number**. The entry number in the record format and based-on file member list. This number then corresponds to the based-on member listed in this entry. This field only applies to logical files and is 0 for a physical file.

| **Maximum number of increments**. The maximum number of increments automatically added to the member size. This field only applies to physical files and is 0 for a logical file.

| **Maximum percentage of deleted records allowed**. The maximum allowed percentage of deleted records for each member in the physical file. The percentage check is made when the member is closed. If the percentage of deleted records is greater than the value shown, a message is sent to the history log. This field only applies to physical files and is 0 when either no deleted records are allowed or the file is a logical file.

| **Member name**. The name of the member whose description is being retrieved.

| **Member text description**. The member's text description.

| **Number of based-on physical file members**. The number of database file members for the logical file member. If the member is a physical file member, the value is 0.

| **Number of days used**. The number of days the member has been used. If the member does not have a last-used date, the value 0 is returned.

| **Number of deleted records**. The number of deleted records returned in the file member. Keyed logical files

return a 0. DDM files that are not from a System/38 or AS/400 system return a 0.

ODP sharing. Whether the open data path (ODP) allows sharing with other programs in the same job. Possible values are:

0 ODP sharing is not allowed. A distributed data management (DDM) file that is sent to a system other than a System/38 or AS/400 system returns a 0.
1 ODP sharing is allowed.

Record capacity. The actual number of records this member can contain. To get this value, multiply the increment number of records by the maximum number of increments, and add the initial number of records. This field only applies to a physical file and is 0 for a logical file.

Record format selector library name. The library in which the record format selector program resides. This field is blank for physical files.

Record format and based-on file list. The number of physical file members this logical file member is based on. There is a maximum of 32 entries. A physical file only has one entry. See "Record Format and Based-On File List Entry" on page 9-19 for a list of the fields contained in this list.

Record format selector program name. The name of a record format selector program that is called when the logical file member contains more than one logical record format.

The user-written selector program is called when a record is written to the database file and a record format name is not included in the high-level language (HLL) program. The selector program receives the record as input, determines the record format used, and returns it to the database. This field is blank for physical files.

Records to force a write. The number of inserted, updated, or deleted records that are processed before the records are forced into auxiliary storage. A 0 indicates that records are not forced into auxiliary storage.

Remote file. Whether the file is a remote file. Possible values are:

0 Local file
1 Remote file

Reserved. An ignored field.

Restore date and time. The date and time that the member was last restored. The restore date and time field is in the CYYMMDDHHMMSS format, which is the same as for the change date and time field. The field contains blanks if the member was never restored. DDM files that are not from a System/38 or AS/400 system return blanks.

Save date and time. The date and time that this member was last saved. The save date and time field is in the CYYMMDDHHMMSS format, which is the same as the change date and time field. This field contains blanks if it was never saved. DDM files that are not from a System/38 or AS/400 system return blanks.

Source file. Whether the file is a source file. The possible values are:

0 Data file
1 Source file

Source type. The type of source member if this is a source file.

SQL file type. The kind of SQL file type the file is. The possible values are:

blank Not an SQL file.
TABLE Nonkeyed physical file that contains field characteristics.
VIEW Logical file over one or more tables or views. This SQL file type provides a subset of data in a particular table or a combination of data from more than one table or view.
INDEX Keyed logical file over one table that is used whenever access to records in a certain order is to be requested frequently.

Use reset date. The century and date when the days-used count was last set to 0. This field is in the CYYMMDD format, which is the same as for the date last used field. If the date is not available, this field is blank.

Error Messages

CPF0600 E Generic messages that could be signalled. These include messages CPF0600 through CPF06FF.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C20 E Error found by program &1.
MCH0000 D All MCHxxxx messages could be signaled. These include messages MCH0000 through MCHFFFF.
CPF3200 D Generic messages that could be signaled. These include messages CPF3200 through CPF32FF.
CPD3C20 D Error occurred with receiver variable specified.
CPD3C21 D Format name &1 is not valid.
CPD3C24 D Length of receiver variable is not valid.
CPD3C25 D Value &1 for file override parameter is not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C22 E Cannot get information about file &1.
CPF3C23 E Object &1 is not a database file.
CPF3C24 E Length of the receiver variable is not valid.
CPF3C25 E Value &1 for file override parameter is not valid.
CPF3C26 E File &1 has no members.
CPF3C27 E Cannot get information about member &3 from file &1.

| CPF3C36 E Number of parameters, &1, entered for this API was not valid.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9804 E Object &2 in library &3 damaged.
| CPF9810 E Library &1 not found.
| CPF9812 E File &1 in library &2 not found.
| CPF9815 E Member &5 file &2 in library &3 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9822 E Not authorized to file &1 in library &2.
| CPF9830 E Cannot assign library &1.

Chapter 10. Commitment Control APIs

The commitment control APIs allow you to add and remove your own resources to be used during AS/400 system commit or rollback processing, or you can retrieve information about the commitment control environment. Commitment control allows you to define and process a number of changes to resources, such as database files or tables, as a single unit (transaction).

Commit or rollback operations include all methods of commit and rollback, such as:

- CL COMMIT command
- PL/I PLICOMMIT subroutine
- SQL COMMIT statement

In this chapter, the terms **commit** and **rollback** indicate all commit and rollback methods applicable on the AS/400 system. A **commitment resource** is any part of the system that is used by a process and placed under commitment control. When a part of the system is put under commitment control by the Add Commitment Resource (QTNADDCR) API described in this chapter, that resource can be referred to as an **API commitment resource**.

API commitment resources are processed during commit, rollback, and process completion but not during an initial program load (IPL).

When commitment control is started, using the Start Commitment Control (STRCMTCTL) command, the system creates a commitment definition, named *BASE, to save internal control information. The commitment definition is maintained by the OS/400 operating system until commitment control is ended. When commitment control is ended, the *BASE commitment definition is destroyed. The *BASE commitment definition includes:

- The parameters on the Start Commitment Control (STRCMTCTL) command
- The current status of the commitment control environment
- Information about files, tables, and other resources that contain changes made during the current transaction

Figure 10-1 on page 10-2 shows how the commitment control APIs can be used together. First, an API used by the high-level language (HLL) program determines if commitment control is active. If it is not active, the HLL then issues the Start Commitment Control (STRCMTCTL) command. Once commitment control is active, another API adds resources to the *BASE commitment definition. As a transaction completes, commit or rollback processing is started. Commitment control calls an exit program for each resource added to the commitment definition. After all transactions are processed, and before commitment control can be ended, another API must remove the resource from the *BASE commitment definition. Finally, the End Commitment Control (ENDCMTCTL) command is issued.

If you plan to use the APIs described in this chapter, you must understand commitment control as discussed in the *Advanced Backup and Recovery Guide*. The commitment control functions using the APIs described in this chapter work the same way as described in the *Advanced Backup and Recovery Guide* with the exception that the API commitment control exit program cannot be used during IPL processing.

The commitment control APIs include the following:

Add Commitment Resource (QTNADDCR) adds an API commitment resource to the *BASE commitment definition.

Remove Commitment Resource (QTNRMVCR) removes an API commitment resource from the *BASE commitment definition.

Retrieve Commitment Information (QTNRCMTI) gets status and lock level information about the *BASE commitment definition.

This chapter presents the APIs in alphabetical order, followed by a description of the commit and rollback exit program.

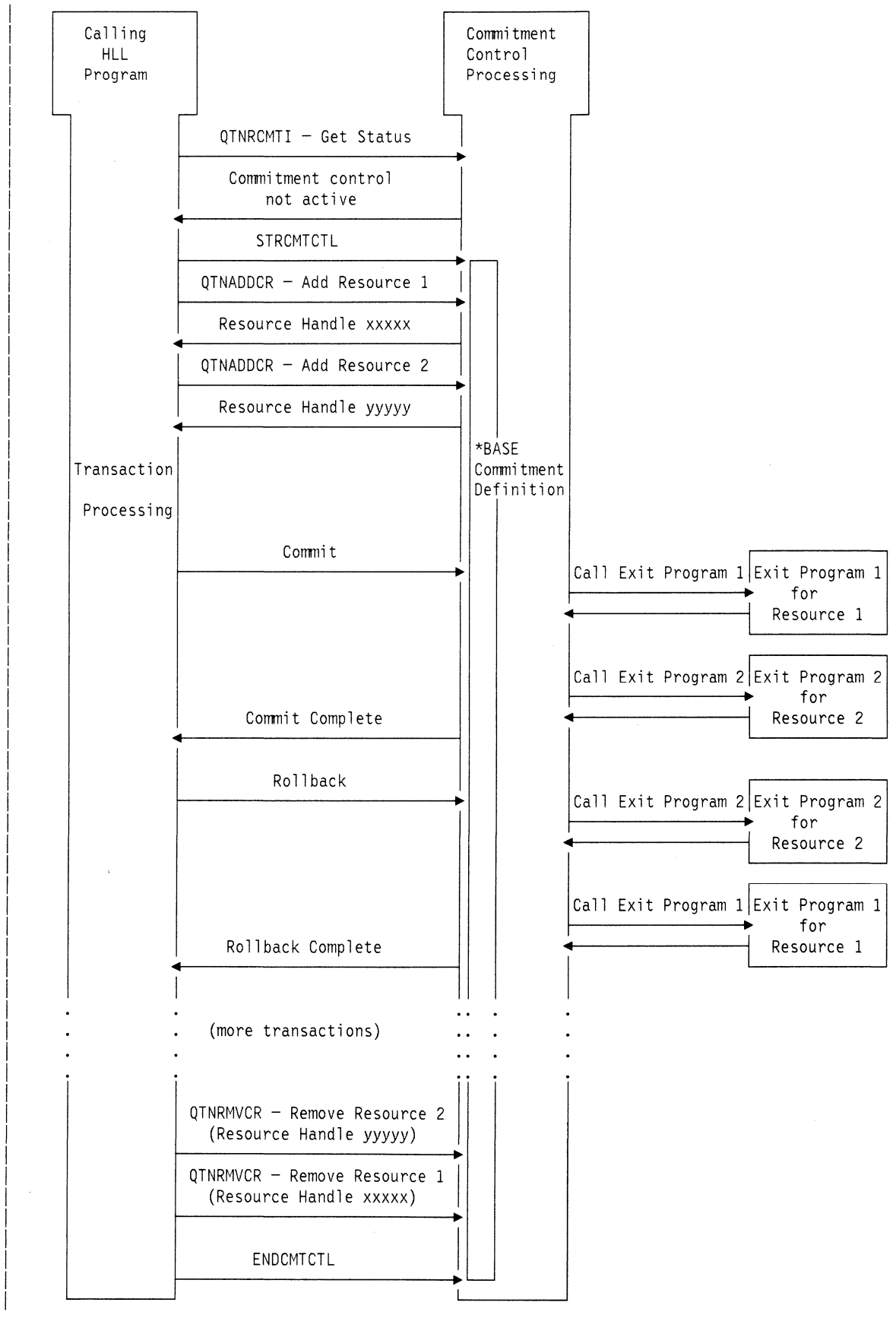


Figure 10-1. Example Using the Commitment Control APIs

Add Commitment Resource (QTNADDCR) API

Parameters

Required Parameter Group:

1	Resource handle	Output	Binary(4)
2	Resource name	Input	Char(10)
3	Qualified commit and rollback exit program name	Input	Char(20)
4	Commit and rollback exit program information	Input	Char(80)
5	IPL processing option	Input	Char(1)
6	Error code	I/O	Char(*)

The Add Commitment Resource (QTNADDCR) API adds an API commitment resource to the *BASE commitment definition. When the resource has been added, the specified exit program is called for each commit or rollback operation performed for the *BASE commitment definition until the resource is removed. The use of this exit program is a form of one-phase commitment control protocol. Once an API commitment resource is added, it must be removed with the Remove Commitment Resource (QTNRMVCR) API before commitment control can end for the *BASE commitment definition. See "Remove Commitment Resource (QTNRMVCR) API" on page 10-4 for more information about this API.

In order to have several API commitment resources at once, you must use this API to add each resource, one at a time. This API does not check for duplicate resource names or duplicate commit and rollback exit programs.

For each API commitment resource that is added, a single call is made to the associated exit program by commit or rollback processing. During commit processing the exit programs are called in the order in which they were added. During rollback processing the exit programs are called in the reverse order. API commitment resource exit programs are called after all record-level I/O operations are processed and are part of the object- or resource-level processing.

Authorities and Locks

Exit Program Authority *USE
Exit Program Library Authority *USE

Required Parameter Group

Resource handle
OUTPUT; BINARY(4)
An identifier made up of an arbitrary number returned by the API and used to identify the commitment resource for subsequent operations, such as the Remove Commitment Resource (QTNRMVCR) API.

Resource name
INPUT; CHAR(10)
The name that identifies this commitment resource. It is used, for example, in some error messages associated with the commit and rollback exit programs.

Qualified commit and rollback exit program name
INPUT; CHAR(20)
The name of the commit and rollback exit program to be called from the commit or rollback operations and the library in which it is located. The first 10 characters of this name contain the program name, and the second 10 characters contain the library name. The special values supported for the library name are *LIBL and *CURLIB.

Note: The special values *LIBL and *CURLIB only apply to the time the resource is added. For example:

1. The API user specifies PROGRAMA in *CURLIB when a commitment resource is added. LIBRARYA is the *CURLIB at the time the resource is added.
2. After the resource addition, *CURLIB is changed to LIBRARYB, which also happens to contain a PROGRAMA.
3. The commit operation occurs and PROGRAMA in LIBRARYA is called, not PROGRAMA in LIBRARYB.

The user of this API must supply this exit program. The considerations for coding this exit program, as well as the information that the commit and rollback operations pass to this exit program, are described in "Commit and Rollback Exit Program" on page 10-6.

Commit and rollback exit program information
INPUT; CHAR(80)
Data to be passed directly to the commit and rollback exit program. This may be any data that is needed by the exit program, such as a reference to an object or area to be used by the exit program. This may be any type of data, including pointers. However, if pointers are used, this field must be on a 16-byte boundary.

Pointers provide better performance than if this parameter were an object name. Resolving to an object on every commit or rollback operation degrades performance. For more information about what information you can expect from a commit and rollback exit program, see "Commit and Rollback Exit Program" on page 10-6.

IPL processing option
INPUT; CHAR(1)
Determines whether the commit and rollback exit program will be called during any commit or rollback processing that occurs during IPL processing.

N The commit and rollback exit program will not be called during the IPL processing phase. This is the only valid value for this option.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Restrictions

You are prevented from adding a commitment resource using this API when:

- Using distributed data management (DDM) remote resources under commitment control for the same process and the *BASE commitment definition[1]
- Using distributed relational database remote resources under commitment control for the same process and the *BASE commitment definition[1]
- Commitment control is not active for the *BASE commitment definition[1]
- Commitment control cannot get a shared-no-update (*SHRNUP) lock on the exit program
- Commit or rollback processing is in progress for the *BASE commitment definition
- The save-while-active function is in progress

Once a resource has been added to the *BASE commitment definition, the process must not change the authorities to or delete the commit and rollback exit program.

Once a resource has been added to the *BASE commitment definition, commitment control cannot be ended until this resource is removed using the QTNRMVCR API. See "Remove Commitment Resource (QTNRMVCR) API" for more information.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF836A E IPL processing option &1 not valid.
CPF836D E Resource name &1 not valid.
CPF8367 E Cannot perform commitment control operation.
CPF8369 E Cannot place API commitment resource under commitment control; reason code &1.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.

Remove Commitment Resource (QTNRMVCR) API

Parameters

Required Parameter Group:

1	Resource handle	Input	Binary(4)
2	Error code	I/O	Char(*)

The Remove Commitment Resource (QTNRMVCR) API removes an API commitment resource that was added to the *BASE commitment definition using the Add Commitment Resource (QTNADDCR) API. For more information, see "Add Commitment Resource (QTNADDCR) API" on page 10-3. Once a commitment resource is removed, the resource handle that refers to it is no longer valid. You cannot end commitment control for the *BASE commitment definition until all API commitment resources have been removed. However, if an End Job (ENDJOB) command is entered or you sign off from the job that is using this commitment definition, commitment control will end and the commitment resources that were added will be removed.

Required Parameter Group

Resource handle
INPUT; BINARY(4)
The resource handle returned by the QTNADDCR API when the API commitment resource was added to the *BASE commitment definition.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Restrictions

You are prevented from removing a commitment resource using this API when:

- The resource handle is not valid.
- Commitment control is not active for the *BASE commitment definition. [1]
- Commit or rollback processing is in progress for the *BASE commitment definition.

In all other instances, the API commitment resource are removed.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF8362 E Cannot remove resource; reason code &1.
CPF8367 E Cannot perform commitment control operation.

[1] You can use the Retrieve Commitment Information (QTNRCMTI) API to retrieve information about what commitment control resources are currently active. See "Retrieve Commitment Information (QTNRCMTI) API" on page 10-5 for more information.

Retrieve Commitment Information (QTNRCMTI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Format name	Input	Char(8)
4	Error code	I/O	Char(*)

The Retrieve Commitment Information (QTNRCMTI) API allows you to determine if a *BASE commitment definition is active and, if it is, what its default lock level is. Information about other commitment definitions is not available through this API.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The receiver variable that is to receive the information requested. You can specify the size of the area smaller than the format requested as long as you specify the receiver variable length parameter correctly. As a result, the API returns only the data the area can hold.

Receiver variable length
INPUT; BINARY(4)
The length of the receiver variable. The length must be at least 8 bytes. If the variable is not long enough to hold the information, the data is truncated. If the length is larger than the size of the receiver variable, the results beyond the length of the largest format are not predictable.

Format name
INPUT; CHAR(8)
The format name CMTI0100 is the only valid format name used by this API. For more information, see "CMTI0100 Format."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

CMTI0100 Format

The structure of the information returned is determined by the specified format name. For detailed descriptions of the fields, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(1)	Commitment definition status
9	9	CHAR(10)	Commitment definition default lock level

Field Descriptions

Bytes available. The length in bytes of all data available to return. All available data is returned if enough space is provided.

Bytes returned. The length in bytes of all data actually returned.

Commitment definition default lock level. The default lock level for the *BASE commitment definition. This is the level of record locking that applies to records in all commitment resources under commitment control for the *BASE commitment definition. The level of record locking is this value unless it is overridden when the file is opened. You cannot override this value; only the system can override for files opened for system functions. Possible values are:

blank Commitment control is not active for the *BASE commitment definition.
**ALL* Lock level *ALL was specified at the start of commitment control for the *BASE commitment definition. Both changed and retrieved records are protected from changes by other jobs running at the same time.
**CHG* Lock level *CHG was specified at the start of commitment control for the *BASE commitment definition. Changed records are protected from changes by other jobs running at the same time.
**CS* Lock level *CS was specified at the start of commitment control for the *BASE commitment definition. Both changed and retrieved records are protected from changes by other jobs running at the same time. Retrieved records are protected only until they are released, or a different record is retrieved.

Commitment definition status. The status for the *BASE commitment definition. Possible values are:

I Commitment control is not active for the *BASE commitment definition.
A Commitment control is active for the *BASE commitment definition, but no local, remote, or API commitment resource is part of the *BASE commitment definition.
L Commitment control is active on the local system for the *BASE commitment definition. Any of the following causes an *L* to be returned:
- One or more local, open database files with pending changes are under commitment control.
- One or more local, closed database files with pending changes are under commitment control.

- One or more resources with object level changes are under commitment control.
- One or more local, distributed relational database resources are under commitment control.
- One or more API commitment resources are part of the *BASE commitment definition.

R Commitment control is active on a remote system for the *BASE commitment definition. Any of the following causes an *R* to be returned:

- One or more remote, DDM resources are under commitment control.
- One or more remote, distributed relational database resources are under commitment control.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF3CF1 E Error code parameter not valid.

Commit and Rollback Exit Program

Users who add API commitment resources to the *BASE commitment definition must supply a commit and rollback exit program as described in the qualified commit and rollback exit program name parameter on page 10-3. The commit and rollback operations call this exit program after all the necessary commit or rollback processing of record-level I/O operations has completed.

The commit and rollback operations pass specific information to the commit and rollback exit program. The exit program must be coded to handle this specific information as described in "Required Parameter Group."

Parameters

Required Parameter Group:

1	Commit and rollback exit program information	Input	Char(80)
2	Status information	Input	Char(32)

Required Parameter Group

Commit and rollback exit program information
INPUT; CHAR(80)
Information associated with the commit and rollback exit program specified when the API commitment resource was added to the *BASE commitment definition. This information is passed to the exit program exactly as it was entered when the API commitment resource was added. The area may contain any data such as pointers or an object name. If pointers are used, each one must start on a 16-byte boundary. A pointer may refer to an area of storage that contains information required by your exit program.

Status information
INPUT; CHAR(32)
Status information from either the commit or rollback operations. Each field of this information has a specific meaning. The fields, their meanings, and size are shown in "Status Information Format."

Status Information Format

The following table shows the offsets, type, and name for the fields passed to the exit program as status information. See "Field Descriptions" for a description of each of these fields.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Status information length
4	4	CHAR(1)	Commit or rollback
5	5	CHAR(1)	Called during IPL
6	6	CHAR(4)	Reserved
10	A	CHAR(1)	Error status
11	B	CHAR(1)	Process end status
12	C	CHAR(20)	Reserved

Field Descriptions

Called during IPL. Whether the exit program was called during the IPL processing phase. This field is set to:

N Not called during IPL processing phase

Commit or rollback. Whether the commit or rollback operation called the exit program. The possible values are:

C Commit operation is calling the exit program
R Rollback operation is calling the exit program

Error status. Whether errors occurred in the commit or rollback processing for this transaction prior to this call to the exit program. The possible values are:

0 No errors occurred
1 Errors occurred

Process end status. Indicates what type of process end called the exit program. The possible values are:

0 Not during the End Commitment Control (ENDCMTCTL) command or any other process end function
1 Normal process end; job ended with a zero completion code
2 Abnormal process end; job ended with a completion code that is not zero

Reserved. An ignored field.

Status information length. The length in bytes of all data passed to the commit and rollback exit program. This field is always set to 32.

Exit Program Locks

OS/400 commitment control obtains a shared-no-update (*SHRNUP) lock on the exit program when the commitment resource is added using the Add Commitment Resource (QTNADDCR) API. This lock is maintained until the resource is removed using the Remove Commitment Resource (QTNRMVCR) API. This locking is done to prevent any changes by other processes to the commit and rollback exit program from the time commitment resources are added until they are used or removed. Changes by other processes, such as deletion, modification, or authority changes, are prevented.

Exit Program Coding Guidelines

When coding a commit and rollback exit program, consider the items in the following lists.

Your exit program **must**:

- Complete its processing within 5 minutes. During a process end the AS/400 system does not allow a commit and rollback exit program to run more than 5 minutes. An exit program cannot prevent a process end.
- Only return an exception to a commit or rollback operation if there has been a failure in the exit program. If the exit program signals anything to commitment control, the system assumes there is a failure and a diagnostic message is returned to the calling program.

Your exit program **must not**:

- Call any commit or rollback operations such as the CL COMMIT command or SQL COMMIT statement. If it does, message CPF8367 is returned to the exit program.
- Call either the QTNADDCR API or the QTNRMVCR API. If it does, message CPF8367 is returned to the exit program.
- Open a local database or DDM file member under commitment control. If it does, message CPF432A is returned to the exit program.
- Start commitment control. If it does, message CPF8351 is returned to the exit program.
- End commitment control. If it does, message CPF8367 is returned to the exit program.

Your exit program **should not** attempt any other commitment control functions, such as:

- Record-level I/O for a local database or DDM file member opened under commitment control
- SQL statements under commitment control

If either of these functions are performed, the results are unpredictable and no error messages are issued.

The following items are good guidelines to follow for any program you write. Your program **should**:

- Handle all potential error conditions. (Fault tolerant)
- Prevent the potential for any infinite looping conditions. (Only during process end will the system stop the exit program after 5 minutes.)
- Be relatively short and perform well.

If your exit program changes any of the required parameter values passed to it, these changes are not preserved for any future calls to the exit program.

Process End Processing Guidelines: During process end processing, debug functions are not necessarily available to help debug any exit program problems. The tasks listed below may be performed during the process end processing phase. If any other actions take place, the commit and rollback exit program may not run successfully or the results will be unpredictable.

- Working with physical files not under commitment control, including creation, changing, opening, closing, clearing, and deletion.
- Database input and output operations to files not under commitment control.
- Working with data areas, including creation, changing, retrieving, and deletion.
- Working with data queues, including creation and deletion.
- Working with message queues, including creation, clearing, changing and deletion.

Some examples of things your exit program might not be able to do during process end are:

- Signal any inquiry messages
- Submit any other jobs
- Use or attempt to start any remote communications activities
- Start any subsystems

Part 4. Edit Function APIs

Chapter 11. Edit Function APIs

This application program interface is used to create and use edit masks. An **edit mask** is a byte string that tells the edit machine instruction or the Edit (QECEDT) API how to format a numeric value into a readable character string.

An edit mask can format a numeric value so that languages that cannot directly use machine instructions can now take advantage of this function. The edit mask was previously defined by the Edit Code (EDTCDE) and Edit Word (EDTWRD) keywords in DDS.

An **edit code** is a standard description of how a number should be formatted. There are many standard edit codes defined by the system. Users can define several edit codes the way they want with the use of the Create Edit Description (CRTEDTD) command. For more information, see the description of the Edit Code (EDTCDE) keyword in the *DDS Reference*.

An **edit word** is a user-defined description of how a number should be formatted. An edit word is usually used when one of the standard edit codes or user-defined edit codes is not sufficient for a particular situation. For more information, see the description of the Edit Word (EDTWRD) keyword in the *DDS Reference*.

The edit function APIs are presented in alphabetical order. They are:

Convert Edit Code (QECCVTEC)
Convert Edit Word (QECCVTEW)
Edit (QECEDT)

Convert Edit Code (QECCVTEC) API

Parameters

Required Parameter Group:

1	Edit mask	Output	Char(256)
2	Edit mask length	Output	Binary(4)
3	Receiver variable length	Output	Binary(4)
4	Zero balance fill character	Output	Char(1)
5	Edit code	Input	Char(1)
6	Fill or Floating currency indication	Input	Char(1)
7	Source variable precision	Input	Binary(4)
8	Source variable decimal positions	Input	Binary(4)
9	Error code	I/O	Char(*)

The Convert Edit Code (QECCVTEC) API translates an edit code specification into an edit mask, which is a byte string used to format a numeric value into a readable character string.

Required Parameter Group

Edit mask
OUTPUT; CHAR(256)
Returns the edit mask generated by this call. The actual length of the edit mask is returned in the edit mask length parameter. The area beyond the actual length of the edit mask is filled with hexadecimal zeros.

The value returned to this parameter should be passed to the Edit (QECEDT) API or the edit machine instruction.

Edit mask length
OUTPUT; BINARY(4)
The actual length of the edit mask.

The value returned in this parameter should be passed to the QECEDT API or used to substring the value returned in the edit mask in the edit machine instruction.

Receiver variable length
OUTPUT; BINARY(4)
Returns the length of the output that is produced by the returned edit mask when it is used.

The value returned in this parameter should be passed to the QECEDT API or used to substring the receiver variable in the edit machine instruction.

Zero balance fill character
OUTPUT; CHAR(1)
Indicates how to perform the edit so that zero balance suppression is done correctly for those edit codes that have zero balance suppression.

The value returned in this parameter should be passed to the QECEDT API or used to determine whether zero suppression requires special handling before issuing the edit machine instruction.

Edit code
INPUT; CHAR(1)
The edit code that is to be translated into an edit mask. The valid values are:

A–D
J–Q
Y–Z
1–9

For more information on edit codes, see the *DDS Reference*.

Fill or floating currency indication
INPUT; CHAR(1)
Indicates how the output should be padded on the left. This parameter should be specified as follows:

" "	**Blank fill:** All suppressed zeros are replaced with blanks.
"*"	**Asterisk fill:** All suppressed zeros are replaced with asterisks.
Character	**Blank fill:** The specified character is used as a floating currency symbol and placed to the left of the first nonsuppressed digit. Characters are X'41' to X'FE'.

Note: You can optionally specify asterisk fill or floating currency symbol with edit codes 1 through 4, A through D, and J through Q.

Source variable precision
INPUT; BINARY(4)
The precision of the numeric variable that is edited with the edit mask. Precision is the number of positions before the decimal point. The valid ranges depend on the value specified for the edit code.

Edit Code	Range
Y	3–7
All others	1–31

The precision of the numeric variable depends on its class:

Variable Class	Precision
Packed	The precision for which the variable was declared. For example, PACKED(8,4) has precision 8.
Zoned	The precision for which the variable was declared. For example, ZONED(8,4) has precision 8.
Binary(2)	5
Binary(4)	10

Notes:

1. Some high-level languages limit the maximum precision of packed and zoned numeric variables.
2. Because the precision of the source variable is so important in creating the edit mask, an edit mask can only be used to edit variables of the exact precision.

Source variable decimal positions
INPUT; BINARY(4)
The number of digits that the source variable precision parameter has placed after the decimal point in the edited output. The value must be less than or equal to source variable precision, but greater than 0. The normal value depends on the class of the source variable precision parameter:

Variable Class	Decimal Position
Packed	The number of decimal positions for which the variable was declared. For example, PACKED (8,4) has 4 decimal positions.
Zoned	The number of decimal positions for which the variable was declared. For example, ZONED (8,4) has 4 decimal positions.
Binary(2)	0
Binary(4)	0

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF2620	E	Field longer than integer or fraction mask.
CPF2639	E	Edit mask too large.
CPF27B2	E	Edit code not valid.
CPF27B3	E	Fill/floating currency indication not valid.
CPF27B4	E	Source variable precision not valid.
CPF27B5	E	Source decimal position not valid.
CPF3CF1	E	Error code parameter not valid.
CPF9801	E	Object &2 in library &3 not found.
CPF9802	E	Not authorized to object &2.

Convert Edit Word (QECCVTEW) API

Parameters

Required Parameter Group:

1	Edit mask	Output	Char(256)
2	Edit mask length	Output	Binary(4)
3	Receiver variable length	Output	Binary(4)
4	Edit word	Input	Char(*)
5	Edit word length	Input	Binary(4)
6	Error code	I/O	Char(*)

The Convert Edit Word (QECCVTEW) API translates an edit word specification into an edit mask. This is useful when one of the standard or user-defined edit codes does not provide the editing required.

Required Parameter Group

Edit mask
OUTPUT; CHAR(256)
Returns the edit mask generated by this call. The actual length of the edit mask is returned in the edit mask length parameter. The area beyond the actual length of the edit mask is filled with hexadecimal zeros.

The value returned to this parameter should be passed to the Edit (QECEDT) API or the edit machine instruction.

Edit mask length
OUTPUT; BINARY(4)
Returns the actual length of the edit mask parameter.

The value returned in this parameter should be passed to the QECEDT API or used to substring the value returned in the edit mask in the edit machine instruction.

Receiver variable length
OUTPUT; BINARY(4)
The actual length of the output that is produced by the returned edit mask when it is used.

The value returned in this parameter should be passed to the QECEDT API or used to substring the value returned in the receiver variable in the edit machine instruction.

Edit word
INPUT; CHAR(*)
The edit word is translated into an edit mask. The character in the system value QCURSYM is treated as a currency symbol if it appears in the edit word.

Edit word length
INPUT; BINARY(4)
The actual length of the edit word. The value passed must be from 1 through 256.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF2639 E Edit mask too large.
CPF27B6 E Edit word length not valid.
CPF3CF1 E Error code parameter not valid.

Edit (QECEDT) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Source variable	Input	*
4	Source variable class	Input	Char(10)
5	Source variable precision	Input	Binary(4)
6	Edit mask	Input	Char(*)
7	Edit mask length	Input	Binary(4)
8	Zero balance fill character	Input	Char(1)
9	Error code	I/O	Char(*)

The Edit (QECEDT) API uses an edit mask to transform a numeric from its internal format to a character form suitable for displaying.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
Receives the edited output. The length of this area must be passed in the receiver variable length parameter.

Receiver variable length
INPUT; BINARY(4)
The length of the referenced area by the receiver variable parameter. This value must be greater than 0.

This value was returned in the receiver variable length parameter on the previous call to the Convert Edit Code (QECCVTEC) API or Convert Edit Word (QECCVTEW) API; otherwise, CPF27AF is returned.

Source variable
INPUT; *
The numeric value that is converted. The type is defined by the source variable class parameter and the length is specified in the source variable precision parameter.

Source variable class
INPUT; CHAR(10)
The type of numeric variable passed in the source variable parameter. The types are:

*BINARY
*PACKED
*ZONED

Source variable precision
INPUT; BINARY(4)
The precision of the numeric variable specified in the source variable parameter. The value passed must be from 1 through 31.

Variable Class	Precision
Packed	The precision for which the variable was declared. For example, PACKED(8,4) has precision 8.
Zoned	The precision for which the variable was declared. For example, ZONED(8,4) has precision 8.
Binary(2)	5
Binary(4)	10

Note: Some high-level languages limit the maximum precision of packed and zoned numeric variables.

Edit mask
INPUT; CHAR(*)
The edit mask used for this edit operation. This is the value returned in the edit mask parameter on the call to the QECCVTEC API or QECCVTEW API.

Edit mask length
INPUT; BINARY(4)
The length of the edit mask. The value passed must be from 1 through 256. This is the value returned in the edit mask length parameter on the call to the QECCVTEC API or QECCVTEW API.

Zero balance fill character
INPUT; CHAR(1)
Indicates how to perform the edit operation so that zero balance suppression is done correctly for those edit codes that have zero balance suppression.

If the QECCVTEC API is used to create the edit mask, this should be the value returned in the zero balance fill character parameter; otherwise, unpredictable results may occur.

If the QECCVTEW API is used to create the edit mask, X'00' should be specified for this parameter; otherwise, unpredictable results may occur.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

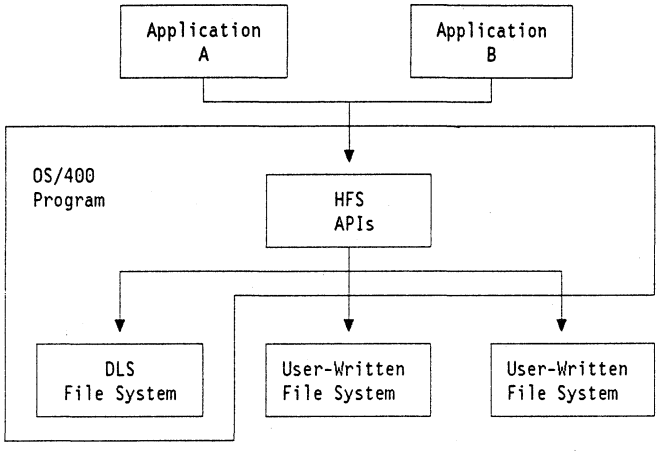
Error Messages

CPF27AB	E	Source variable class not valid.
CPF27AF	E	Edit mask not valid.
CPF27B4	E	Source variable precision not valid.
CPF27B7	E	Receiver variable length not valid.
CPF27B8	E	Edit mask length not valid.
CPF3CF1	E	Edit code parameter not valid.

Part 5. Hierarchical File System APIs

Chapter 12. Introduction to the Hierarchical File System APIs

The hierarchical file system (HFS) APIs and the functions that they support are part of the OS/400 program. The APIs provide applications with a single, consistent interface to all the hierarchical file systems available on your AS/400 system. They automatically support the document library services (DLS) file system and can support user-written file systems also. The following diagram shows the relationship of the HFS APIs to your applications and to other file systems:



The HFS APIs allow you to work with nonrelational data stored in objects, such as directories and files in existing file systems. Using these APIs, you can perform such tasks as creating and deleting directories and files, reading from and writing to files, and changing the directory entry attributes of files and directories.

The APIs allow you to perform a wide variety of tasks in the following categories:

File System Management APIs help you manage your use of file systems in general. The file system management APIs include the following:

List Registered File Systems (QHFLSTFS) lists the file systems that are registered on your AS/400 system and thus available for use through the HFS APIs.
Control File System (QHFCTLFS) allows your applications to issue file-system-specific commands.

In each section, the main text and tables describe how the APIs work in hierarchical file systems in general. Footnotes describe any differences and additional considerations for the document library services (DLS) file system.

Directory Management APIs allow you to perform general maintenance tasks for directories in hierarchical file systems. The APIs include the following:

Create Directory (QHFCRTDR)
Rename Directory (QHFRNMDR)
Delete Directory (QHFDLTDR)

Other APIs, listed in "Directory Entry Information APIs," let you open and close directories and work with their directory entry attributes.

File Input and Output APIs allow you to work with the contents of files in hierarchical file systems. The APIs work with **stream files,** which are files that have varying lengths and no conventional record structure. Stream files are also called byte-stream files, or simply files. The file input and output APIs include the following:

Open Stream File (QHFOPNSF)
Read from Stream File (QHFRDSF)
Write to Stream File (QHFWRTSF)
Lock and Unlock Range in Stream File (QHFLULSF) allows you to lock and unlock parts of files.
Change File Pointer (QHFCHGFP) allows you to change the location of the current read/write position in the file.
Force Buffered Data (QHFFRCSF) forces data from a buffer into nonvolatile storage. (**Nonvolatile storage** is any storage area whose contents are not lost when power is cut off or when the system is loaded.)
Get Stream File Size (QHFGETSZ)
Set Stream File Size (QHFSETSZ)
Close Stream File (QHFCLOSF)

File Management APIs allow you to perform general maintenance tasks for files in hierarchical file systems. The file management APIs include the following:

Copy Stream File (QHFCPYSF)
Move Stream File (QHFMOVSF)
Rename Stream File (QHFRNMSF)
Delete Stream File (QHFDLTSF)

The APIs listed in "File Input and Output APIs," let you read and write data to files. The APIs described in "Directory Entry Information APIs," let you work with the directory entries for files.

Directory Entry Information APIs allow you to work with the directory entries for files and directories in hierarchical file systems. The directory entry information APIs include the following:

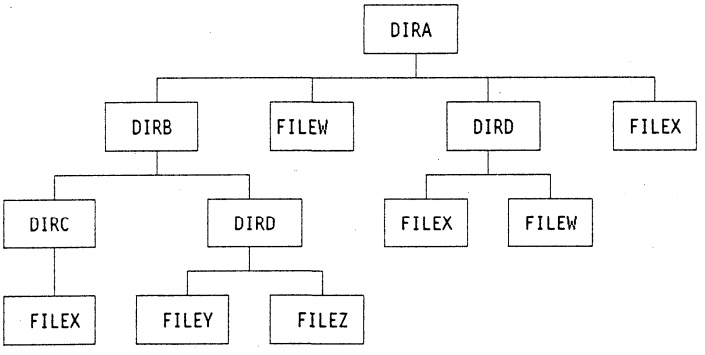
Open Directory (QHFOPNDR)
Read Directory Entry (QHFRDDR)
Retrieve Directory Entry Attributes (QHFRTVAT)
Change Directory Entry Attributes (QHFCHGAT)
Close Directory (QHFCLODR)

Before using the HFS APIs, read the remaining sections in this chapter. They discuss basic HFS concepts, explain authority to use file systems and their objects, and describe directory entry attributes in detail. Next, if you plan to use the HFS APIs to work with the document library services (DLS) file system, read Chapter 13, "The Document Library Services (DLS) File System." If you are not writing applications but instead creating or installing a new file system for other programmers to use with the HFS APIs, turn to Chapter 15, "Preparing to Use New File Systems with the HFS APIs."

HFS Terms and Concepts

The HFS APIs work with units of information or objects that belong to existing, hierarchical file systems. A **file system** is the operating system's method of controlling the format of information on storage media and performing input and output operations to the objects that contain the information. The document library services (DLS) file system is one example of a file system. The basic units of nonrelational information in a file system are usually called **files**. Files are sometimes called **byte-stream files** or **stream files** because they consist of a stream of bytes with no specific record structure. In addition, they are sometimes called **documents** because they can be used for textual items such as letters and reports.

A **hierarchical file system** arranges information units in a multilevel, tree-like structure, as the IBM DOS system does. Files are grouped into larger units usually called directories. A **directory**, sometimes called a **folder**, can contain both files and subordinate directories. A directory contains no data of its own but is simply a named group of files and other directories. The following diagram illustrates the structure of a hierarchical directory:



In the preceding diagram, the topmost directory, DIRA, contains both directories, DIRB and DIRD, and files, FILEW and FILEX. Directory DIRD contains only files. Directory DIRB contains only directories. In this structure, there are two directories named DIRD, one in directory DIRA and one in directory DIRA/DIRB. There are also three files named FILEX, one in directory DIRA, one in DIRA/DIRD, and one in DIRA/DIRB/DIRC. Two objects with the same name cannot exist in the same directory, but one directory can contain directories or files with the same names as those in another directory.

A file's or directory's specific location and name are represented in a multipart name called a path name. A **path name** starts with a slash (/) and consists of elements separated by slashes. The first element of the path name is the name of the file system. The remaining elements specify the applicable directory and file names; the last element can be either a file or a directory, but the rest must be directories. For example, the path name /QDLS/DIRA/FILEW refers to file FILEW in directory DIRA in file system QDLS.

Authority to HFS APIs and File Systems

Before you can use an HFS API to work with a particular file system, these conditions must be met:

1. You must have *USE authority to the API. This gives you the authority to call the API from your programs.
2. You must have authority to use the file system. Authority to the file system is controlled by the file system's job startup program, described in "Start Job Session Exit Program" on page 15-6.[1] If you have authority to use the file system's Start Job exit program, you have authority to the file system as a whole. Authority to use files and directories within a file system is controlled by the file system itself.
3. The HFS API you want to use must be available for use with the file system. Some file systems might not support all the HFS APIs described in this book.

Directory Entry Attributes

Every file and directory in a file system has a corresponding directory entry. The **directory entry** is created automatically when the file or directory is created. It is stored in the directory in which the file or directory is located and contains descriptive information, such as the item's creation date and whether it is a file or directory. The items of information are called **directory entry attributes** or simply **attributes**. Each attribute has a name as well as a value.

Standard Directory Entry Attributes

Some directory entry attributes are created automatically when the directory entry is created. These attributes are called **standard attributes**. Their names start with the letter Q so you can identify them easily. The file system determines which standard attributes you can specify when working with directory entries.

The standard attributes for HFS directory entries are:

QNAME
CHAR(*)
The current name of a file or directory.

Do not specify this attribute in the attribute information table or attribute selection table used with some APIs. The name is either already specified or disallowed:

- When you create a directory or file and when you retrieve directory entry attributes, the object's name is already specified in the API's path name parameter.

[1] Authority to use the document library services (DLS) file system is handled differently. See "User Enrollment" on page 13-2.

- The Read Directory Entries (QHFRDDR) API always returns the QNAME attribute to identify the name of the directory entry.
- You cannot use the Change Directory Entry Attributes (QHFCHGAT) API to change this attribute and rename the object. You must use the Rename Directory (QHFRNMDR) or Rename Stream File (QHFRNMSF) API to rename the object.

QFILSIZE
BINARY(4) UNSIGNED
The size of a file's data, in bytes.

This attribute applies only to files. Thus, for directories, it has a value of zero. If you specify it for a directory, it is ignored.

QALCSIZE
BINARY(4) UNSIGNED
For a file, the allocated size of the file in bytes. The allocated size is the amount of space the file system actually uses to store the file.

This attribute applies only to files. Thus, for directories, it has a value of zero. If you specify it for a directory, it is ignored.

QCRTDTTM
CHAR(13)
The date and time of day the file or directory was created, in CYYMMDDHHMMSS format.

You cannot specify this attribute when creating a directory or file. However, you can specify it on calls to the Retrieve Directory Entry Attributes (QHFRTVAT) and Change Directory Entry Attributes (QHFCHGAT) APIs.

QACCDTTM
CHAR(13)
The date and time of day the file or directory was last accessed, in CYYMMDDHHMMSS format.

Note: The DLS file system does not support the file access date and time. If you specify the QACCDTTM attribute, it is accepted but ignored.

You can specify this attribute on any API call, but your file system might ignore it in on some APIs.

QWRTDTTM
CHAR(13)
The date and time of day the file or directory was last written to, in CYYMMDDHHMMSS format.

You can specify this attribute on any API call, but your file system might ignore it in on some APIs.

QFILATTR
CHAR(10)
The type of item the directory entry is for. You can specify this attribute on any API call, except as noted in the following list.

Each character in the QFILATTR attribute has a specific meaning. Characters 6-10 must be set to blanks. Characters 1-5 must have a value of either 0 or 1:

0 No
1 Yes

The characters and their meanings are:

1 Read-only file. This applies only to files; it is ignored for directories. You can change it only by using the Change Directory Entry Attributes (QHFCHGAT) API.

When a file has this attribute, the file cannot be accessed in write mode. It cannot be opened with write or read/write access, it cannot be the target file in a copy stream file operation, and it cannot be deleted.

2 Hidden file or directory. You can change this attribute by using the Change Directory Entry Attributes (QHFCHGAT) API.

3 System file or directory. You can change this attribute by using the Change Directory Entry Attributes (QHFCHGAT) API.

4 Entry is a directory (not a file). You cannot change this attribute. If you specify it on the Change Directory Entry Attributes (QHFCHGAT) API, it is ignored.

5 Changed file. This applies only to files. It indicates that the file has been changed and is usually used to determine when a file needs to be moved to safe, permanent storage. It is set to Yes when the file is created or written to. You can set it to No only by using the Change Directory Entry Attributes (QHFCHGAT) API.

6-10 Reserved. Must be set to blanks.

QERROR
CHAR(7)
A special attribute that can be returned in the attribute data buffer by the Read Directory (QHFRDDR) API when it encounters an error in retrieving the attributes of a directory entry. These values can be returned:

CPF1F06	Directory in use.
CPF1F08	Damaged directory.
CPF1F26	File in use.
CPF1F28	Damaged file.
CPF1F62	Requested function failed.
CPF1F71	File system unique exception occurred.

For details about calling the QHFRDDR API from an application, see page 14-16. For details about the interface between the QHFRDDR API and a new file system, see page 15-22.

Other Directory Entry Attributes

Files and directories can have attributes other than the standard ones. For example, file systems can define their own unique attributes. The file system must define the attributes' names and values, and the file system doc-

umentation must tell users how to access and use them.[2]

An application can also define its own directory entry attributes. These attributes are sometimes called **extended attributes.** They resemble the extended attributes in the IBM OS/2* file system and supply additional information relevant to the application. The application must define the names and values of these attributes.

Attribute Information Table

The HFS APIs use a common attribute information table to pass all types of directory entry attributes between the application and the file system. Thus, the information is returned in the same format regardless of which file system the application is using. The table consists of zero or one or more attributes and varies in length. Different file systems can use different attributes, so the contents of the table can vary from one file system to another.

The attribute information table is used by these HFS APIs:

Create Directory (QHFCRTDR)
Retrieve Directory Attributes (QHFRTVAT)
Change Directory Entry Attributes (QHFCHGAT)
Read Directory Entries (QHFRDDR)
Open Stream File (QHFOPNSF)

The attribute information table has three logical parts:

1. The first field specifies the number of attributes defined in the table.
2. The next fields give the offsets to the attributes defined in the table. There is one offset field for each attribute.
3. Next are groups of fields describing the attributes being defined or retrieved. There is one group of descriptive fields for each attribute.

The format of the attribute information table is:

Type	Field
BINARY(4)	The number of attributes defined in the table. This is the number of attributes being defined for or retrieved from the directory entry.
Offsets:	
BINARY(4)	The offset to the first attribute.
BINARY(4)	The offset to the next attribute, if more than one is being defined or retrieved. This field is repeated to list the offset to each attribute being defined or retrieved.
Description of the first attribute:	
BINARY(4)	Length of attribute name
BINARY(4)	Length of attribute value
BINARY(4)	Reserved; currently set to zero
CHAR(*)	Attribute name
CHAR(*)	Attribute value
Description of the next attribute (repeated for each attribute after the first):	
BINARY(4)	Length of attribute name
BINARY(4)	Length of attribute value
BINARY(4)	Reserved; currently set to zero
CHAR(*)	Attribute name
CHAR(*)	Attribute value

Attribute Selection Table

The HFS APIs use a common attribute selection table to choose which directory entry attributes to retrieve or to make available for reading. The attribute selection table varies in length.

The attribute selection table is used by these HFS APIs:

Open Directory (QHFOPNDR)
Retrieve Directory Entry Attributes (QHFRTVAT)

The attribute information table contains an entry for every attribute the application selects. If a selected attribute does not exist for the directory entry, no error is signaled. The attribute name is returned, and the length of the attribute value is zero.

Like the attribute information table, the attribute selection table has three logical parts:

1. The first field specifies the number of attributes specified in the table.
2. The next fields give the offsets to the attributes specified in the table. There is one offset field for each attribute.
3. Next are pairs of fields describing the attributes specified. There is one pair of descriptive fields for each attribute.

The format of the attribute selection table is:

Type	Field
BINARY(4)	The number of attributes specified in this table
Offsets:	
BINARY(4)	The offset to the first attribute

[2] The document library services (DLS) file system has several attributes of its own. It also places restrictions on the standard attributes you can use when working with DLS files and directories. For details, see "Attributes to Use When Creating Directories and Files" on page 13-4.

Type	Field
BINARY(4)	The offset to the next attribute, if more than one is specified. This field is repeated for each attribute.
Description of the first attribute:	
BINARY(4)	Length of attribute name
CHAR(*)	Attribute name
Description of the next attribute (repeated for each attribute specified):	
BINARY(4)	Length of attribute name
CHAR(*)	Attribute name

Chapter 13. The Document Library Services (DLS) File System

The document library services (DLS) file system is a file system used with the OfficeVision/400* licensed program. You can use the HFS APIs to work with DLS directories and files. This chapter describes the parts of the DLS file system that are especially relevant to applications. It covers these topics:

- Terms and concepts that differ from those used in other hierarchical file systems
- User enrollment
- Authorities needed to work with DLS objects
- CL commands you can use to work with DLS objects
- Commitment control
- Directory entry attributes

Before working with the DLS file system, you should also review these publications:

- *Office Services Concepts and Programmer's Guide* describes basic DLS file system maintenance and use. It explains how OfficeVision/400 users create and access DLS objects and describes the user commands that resemble the HFS APIs in function.
- *Advanced Backup and Recovery Guide* describes the backup procedures for the DLS file system in detail.

Terms and Concepts

The primary components of the DLS file system are directories and files, just as in other hierarchical file systems. However, the DLS file system uses several special terms to refer to these objects:

Figure 13-1. DLS Terminology

Common Term	DLS Term
File	Document A DLS document can contain up to 2 147 483 647 bytes. DLS document (*DOC) objects differ from OS/400 file (*FILE) objects. The former are byte-stream files like those in other hierarchical file systems. The latter are system database files.
Directory	Folder A DLS folder can contain up to 65 535 folders and documents. As the number of objects in a folder increases, the time to access the objects also increases, especially when deleting and inserting objects. For best performance, limit folders to 1000 objects.
File or directory	Document library object (DLO). This is a collective term for files and directories in the DLS file system.
Directory entry attribute	Directory entry attribute or interchange document profile (IDP) parameter. Some DLS attributes are IDP parameters; others are ordinary directory entry attributes.

The following sections describe DLS file and directory naming conventions, file types, and the ways in which users can manipulate DLS files and directories.

File and Directory Names

The *CL Reference* includes a complete discussion of the rules for specifying DLS file (document) and directory (folder) names. In general, path names for the DLS file system have the same format as path names for ordinary hierarchical file systems, described in "HFS Terms and Concepts" on page 12-2. In the DLS file system, any name in the path can have two parts, the main part and an optional suffix. The main part of the name can be from 1 to 8 characters long. The optional suffix must start with a period and have 1 to 3 additional characters.

DLS file and directory names can contain almost any value in which the value of each character is greater than hex 40 and less than hex FF. Hex 40, hex FF, and the following characters are not valid:

Hex 5C Asterisk
Hex 61 Slash
Hex 6F Question mark

A period (hex 4B) can be used only as the first character of the optional suffix. Lowercase letters a-z are changed to uppercase letters A-Z. Thus, the name that the system stores does not contain characters hex 81 through hex 89, hex 91 through hex 99, or hex A2 through hex A9.

The total length of a DLS path name cannot exceed 82 characters, including the file system name and all leading and separating slashes (/). If the path name specifies a directory rather than a file, it cannot exceed 69 characters.

File Types

DLS file system users can create files (documents) with data of different types. Some types, such as those within Document Content Architecture, are registered with Document Interchange Architecture (DIA). System users must be careful not to alter these types. For example, if a user creates a document of a type that other users expect to be in DIA form but that does not have DIA form, unpredictable results can occur. Users might not be able to edit or otherwise process the file any more. For further details about DLS document types, see the DOCTYPE parameter of the File Document (FILDOC) command in the *CL Reference*.

Manipulating Files and Directories

Users can manipulate DLS files (documents) and directories (folders) in several ways. For example, a file (document) created with the Open Stream File (QHFOPNSF) API can be deleted later with the Delete Document Library Object (DLTDLO) command. A directory (folder) can be created with the Create Folder (CRTFLR) command, and that folder can be manipulated with directory APIs. "CL

Commands for DLS Users" on page 13-2 lists the CL commands you can use to work with DLS file system objects. Users can also work with documents and folders through the PC Support/400 and OfficeVision/400 licensed programs.

User Enrollment

Before users can work with the DLS file system, they must be enrolled in the system distribution directory. To enroll and remove users, use the OfficeVision/400 administration option, described in *Managing OfficeVision/400*, or the Add Directory Entry (ADDDIRE), Work with Directory (WRKDIR), and Remove Directory Entry (RMVDIRE) commands, described in the *CL Reference*.

Object Authorities

The DLS file system uses an object authority system that closely resembles the regular AS/400 authority system. The major differences are as follows:

- Normally, the user needs authority to an AS/400 library to use objects in the library. In the DLS file system, however, the user does not need authority to the object that contains the object being accessed.
- The CL commands normally used to work with system object authorities are not valid for DLS file system objects. Instead, the user must use commands specific to DLS file system objects, such as the Add Document Library Object Authority (ADDDLOAUT) command.
- Authority is always inherited from the containing object. For example, when a file (document) is created, it takes its authority specifications from the directory (folder) in which it is located.

Security Reference describes AS/400 system and object authority in detail. *Office Services Concepts and Programmer's Guide* provides more information about DLS security.

In the DLS file system, the four levels of object authority are:

***EXCLUDE**	The user cannot access the object in any way.
***USE**	The user can view and retrieve data from the object but cannot change or delete it.
***CHANGE**	The user can view, retrieve data from, and change the object but cannot delete it.
***ALL**	The user can view, retrieve data from, change, and delete the object. The owner of an object does not always have this level of authority.

The authority can be set for a specific user in the object, as public authority in the object, in an authorization list, or in a group profile. The authority for a directory or file can be changed with CL commands such as Add Document Library Object Authority (ADDDLOAUT) and Change Document Library Object Authority (CHGDLOAUT).

When a new file or directory is created, it is automatically given the authority of the directory in which it is contained. When a DLS directory is created at the topmost level of the file system (that is, in the file system itself and not within any other directory), its public authority is *EXCLUDE.

Users do not need authority to a directory to access the files and directories within it. They need authority only to the specific objects they want to use. For example, to retrieve directory entry attributes of files or directories, the user must have at least *USE authority to the files or directories. If a user has *USE authority to some objects in a directory but no authority to others, the system returns the attributes for only the authorized objects. A user can have the authority to access a file without having authority to read the directory entry attributes of the directory in which the file is located.

The authorities needed to use the HFS APIs with the DLS file system are noted in the HFS API chapters following this chapter.

CL Commands for DLS Users

The following list shows the commands that you can use to work with DLS files (documents) and directories (folders). These commands are fully described in the *CL Reference*. You should not use any other CL commands to manipulate DLS files and directories. However, there are OfficeVision/400 commands that you can use with DLS file system objects; see the *Office Services Concepts and Programmer's Guide* for details.

Some of the commands in the following list assume that the document is in a particular form. For example, the Edit Document (EDTDOC) command assumes that the document is in a form that can be edited using OS/400 word processing.

These are commands you can use with DLS files and directories, which are collectively called document library objects (DLOs):

ADDDLOAUT	Add Document Library Object Authority
CHGDLOAUT	Change Document Library Object Authority
CHGDLOOWN	Change Document Library Object Owner
CHKDLO	Check Document Library Object
DLTDLO	Delete Document Library Object
DSPDLOAUT	Display Document Library Object Authority
EDTDLOAUT	Edit Document Library Object Authority
RCLDLO	Reclaim Document Library Object. You can use this command to repair a document that the system reports as being in an inconsistent state when you attempt to access it.
RGZDLO	Reorganize Document Library Object
RMVDLOAUT	Remove Document Library Object Authority
RNMDLO	Rename Document Library Object
RSTDLO	Restore Document Library Object

SAVDLO	Save Document Library Object. You can use this command to make a copy of one or a group of objects on offline media for safe, permanent storage. The Restore Document Library Object (RSTDLO) command above restores the object.
CHGDOCD	Change Document Description
CHKDOC	Check Document
CPYDOC	Copy Document
CRTDOC	Create Document
DLTDOC	Delete Document
DSPDOC	Display Document
EDTDOC	Edit Document
FILDOC	File Document
MOVDOC	Move Document
PRTDOC	Print Document
QRYDOCLIB	Query Document Library
RPLDOC	Replace Document
RTVDOC	Retrieve Document
SNDDOC	Send Document
WRKDOC	Work with Document
WRKDOCLIB	Work with Document Library
WRKDOCPRTQ	Work with Document Print Queue
CRTFLR	Create Folder
DSPFLR	Display Folder
WRKFLR	Work with Folder

Commitment Control

DLS file system users can make only limited use of commitment control because the system uses commitment control when creating, deleting, and changing DLS documents (files) and folders (directories). **Commitment control** is a means of grouping file operations so that a group of changes can be committed or removed as a single unit. For a discussion of how to use commitment control and journaling, see the *Office Services Concepts and Programmer's Guide*.

Directory Entry Attributes

The following sections discuss these directory entry attributes:

- Standard attributes
- Interchange document profile (IDP) and other attributes
- Extended attributes
- Attributes you can use when creating directories and files
- Retrieving attributes

Standard Attributes

You can use these standard directory entry attributes for DLS directories and files:

- QALCSIZE
- QCRTDTTM
- QFILATTR
- QFILSIZE
- QNAME
- QWRTDTTM

These attributes are described in detail in "Standard Directory Entry Attributes" on page 12-2.

If you misspell a standard attribute name, the DLS file system handles the attribute as an extended attribute, which in most cases would not produce meaningful results.

Interchange Document Profile (IDP) and Other Attributes

Except for the standard attributes, all directory entry attributes for DLS files and directories are stored in the interchange document profile (IDP). The IDP is defined by the Document Interchange Architecture (DIA) and is a part of each DLS object that stores information about the object. The IDP is separate from the object's data and is divided into logical parts called subprofiles. The IDP contains both extended attributes, which are not defined by DIA, and DIA-defined attributes. Directory entry attributes stored in the IDP are sometimes called **IDP parameters**. For more information about the IDP and DIA, see the *IDP Reference*.

Attribute Names: When specifying the names of attributes stored in the IDP in calls to HFS APIs, applications must use the attribute format used by the IDP. Attributes are stored in the IDP as fields. An IDP field can contain a DIA-defined attribute or an extended attribute. All DIA-defined IDP fields can be accessed by specifying a name in the form DIA.*ssssssss*, where *ssss* is the character representation of the hexadecimal subprofile identification and *pppp* is the character representation of the hexadecimal code point of the field as it is identified in the IDP. For example, the base subprofile has an ID of hex CA04. The author field in the base subprofile has a code point of hex C704. You can retrieve the author field by requesting the field named DIA.CA04C704 as an extended attribute.

Attribute names in this form must be exactly 12 characters long.

The profile graphic code-page identifier (GCID) has a code point of hex C701 and is also in the base subprofile. You can retrieve it by requesting the attribute named DIA.CA04C701. The profile GCID is set at file creation time and it cannot be updated.

Any attribute name that has the letters DIA and a period (`DIA.`) as the first 4 characters is treated by the system as a DIA attribute name and must be in the form described here and in the *IDP Reference*. If the name and data are not specified in this form, an error is returned.

Some standard attributes can also be accessed using this form of specification. When this is done, the data returned might be in a different form from the data returned when the attribute is specified as a standard attribute. For example, the date and time attributes are returned in DIA internal form, not in the form specified for standard attributes.

As output from a retrieve operation or as input to an update operation performed with an HFS API, the name

and length of the field are set in the API. The length of the name and the length of the data are set or returned as separate parameters. This differs slightly from the data structures defined in the *IDP Reference*.

Attribute Values: When applications pass attribute values to or receive attribute values from the system in the attribute information and selection tables, the values are in the following form:

The first field is the hexadecimal code point of the attribute (CP).

The second field is the format type (F). It can be either hex 01, indicating that this is a single-parameter structure or hex 41, indicating that this is a parameter-set structure.

For example, for the author field either of the following forms could apply:

1. Single parameter structure (format 01). This form could be used when the only subfield that is defined is the author name:

```
  CP   F
|C704|x'01'|author|
```

2. Parameter set structure (format 41). This form could be used when a profile GCID and other data are associated with the author name:

```
  CP   F    L    T         L    T          L    T
|C704|x'41'|x'06'|x'01'|gcid|x'04'|x'14'|lgrpid|x'nn'|x'15'|ctryid|

   L    T          L    T    L    T
 x'nn'|x'16'|nlngid|x'02'|x'05'|x'nn'|x'02'|author|

   L    T       L    T
 x'nn'|x'03'|dgn|x'nn'|x'04'|den|...
```

   The length field (L) contains the length of the length, type, and data fields.

   The type field (T) contains the type of data that follows. The meaning of each type can be unique for each code point.

   Not all of the fields shown are required, and not all available fields are shown. In this parameter the fields can be repeated to describe more than one author. Refer to the *IDP Reference* for complete information.

For the profile GCID, the following form (format 01) is the only acceptable form:

```
  CP   F
|C701|x'01'|gcid|
```

When using the parameter set structure (format 41), the presence or absence of any given field is not assured. The data returned is exactly what is stored in the profile and can vary. For example, there might be one, more than one, or no explicit GCIDs for the fields. If a field requires a GCID but does not have an explicit GCID, it uses the profile GCID.

Attributes with Multiple Values: When an IDP attribute is updated, the entire attribute is replaced. When you are using the HFS APIs, you cannot add or replace one value of a multiple-value attribute. For example, you cannot add or remove an author to an existing list of authors, but you can retrieve the author attribute, change the attribute contents, and replace the attribute. The *IDP Reference* describes the format of all profile parameters.

Extended Attributes

Attributes that are specified without the `DIA.` prefix are stored in the extended attribute subprofile. You can request them by specifying only the parameter name and data (with appropriate length fields supplied). The name can be any character string up to 128 characters long that does not begin with the characters `DIA.` (the letters DIA plus a period). In addition, to avoid confusion with standard attributes, you should not give extended attributes names beginning with the letter Q. The data can be in any form chosen by the user.

Attributes to Use When Creating Directories and Files

When creating a directory (folder) in the DLS file system, you can specify two file-system-defined attributes: text description and profile GCID. When creating or replacing a file (document), you can specify those two attributes and a third, file type. The three attributes are described below:

File type

(The file's document type) This attribute can have many different values; for a list, see the *IDP Reference*. The attribute name is DIA.CA04C706. The attribute data must be in the following form, where *type* is the 2-byte value of the document type desired:

x'C706'	x'01'	type

The length-of-attribute-value field specified in the API's attribute information table must be 5. See "Attribute Information Table" on page 12-4. If this attribute is not provided when a file is opened, a value of hex 000E is assumed. This value is the document type for an IBM Personal Computer file.

You cannot change this attribute with the Change Directory Entry Attributes (QHFCHGAT) API.

Profile GCID

(Profile graphic code-page identifier) The GCID is a code that identifies which graphic elements, such as letters and numbers, are associated with any value in a character. The GCID is usually associated with a language; sometimes each country that uses a given language uses a separate GCID. The GCID identifies the code page in which the data is stored and can be used in translating text data from one code page to another. Sometimes called the character ID (CHRID) instead, the GCID is also used to translate keyboard

input into a common code page when creating AS/400 display files.

The profile GCID identifies the code page of the attribute data, for attributes that do not have an explicit GCID associated with them. You can specify the profile GCID attribute when creating or replacing a file and when creating a directory. The attribute name is DIA.CA04C701. The attribute data must be in the following format, where *gcid* is the 4-byte value of the graphic code-page identifier for the profile:

x'C701'	x'01'	gcid

The length-of-attribute-value field specified in the API's attribute information table (see "Attribute Information Table" on page 12-4) must be 7. If this attribute is not provided, a profile GCID of code page 697 and character set 500 is used.

You cannot change this attribute with the Change Directory Entry Attributes (QHFCHGAT) API.

Text description

An internal document name 1 to 44 characters long. The attribute name is DIA.CA04C700. The attribute data must be in the following format, where *text* is the character data describing the directory entry:

x'C700'	x'01'	text

The length-of-attribute-value field specified in the attribute information table (see "Attribute Information Table" on page 12-4) must be the length of the text string, plus 3 for the fields preceding the text. The text must be encoded in the character set and code page specified by the GCID.

If you do not specify this field when creating a file or directory, the file system uses the name of the file or directory being created as the value.

You can change this attribute with the Change Directory Entry Attributes (QHFCHGAT) API, but you cannot delete it.

Retrieving Attributes

When you retrieve directory entry attributes in the DLS file system using the Read Directory Entries (QHFRDDR) API, the returned data might appear to be incomplete or incorrect. This is because of the manner in which the DLS file system performs open, read, and delete operations. When a user opens a DLS folder (directory), the system records the folders and documents (files) that are in that directory at that time. After the open operation:

- If more folders or documents are added to the directory before the read operation, they do not appear in the data returned to the user.
- If folders or documents have been deleted from the directory, they might or might not appear in the output from the read operation:
  - If the user requests only standard directory entry attributes, the deleted objects might appear in the output.
  - If the user requests additional attributes that must be extracted from the object, the deleted objects are ignored and no error is returned to the user.

Chapter 14. Hierarchical File System APIs

This chapter describes the HFS APIs that let you work with files and directories and with the data stored in files. The APIs in this chapter are presented in alphabetical order.

For each API description, the main text describes how the API works in hierarchical file systems in general. Footnotes describe any differences and additional considerations for the document library services (DLS) file system.

The HFS APIs include the following:

Change Directory Entry Attributes (QHFCHGAT)
Change File Pointer (QHFCHGFP) allows you to change the location of the current read/write position in the file.
Close Directory (QHFCLODR)
Close Stream File (QHFCLOSF)
Control File System (QHFCTLFS) allows your applications to issue file-system-specific commands.
Copy Stream File (QHFCPYSF)
Create Directory (QHFCRTDR)
Delete Directory (QHFDLTDR)
Delete Stream File (QHFDLTSF)
Force Buffered Data (QHFFRCSF) forces data from a buffer into nonvolatile storage. (**Nonvolatile storage** is any storage area whose contents are not lost when power is cut off or when the system is loaded.)
Get Stream File Size (QHFGETSZ)
List Registered File Systems (QHFLSTFS) lists the file systems that are registered on your AS/400 system and thus available for use through the HFS APIs.
Lock and Unlock Range in Stream File (QHFLULSF) allows you to lock and unlock parts of files.
Move Stream File (QHFMOVSF)
Open Directory (QHFOPNDR)
Open Stream File (QHFOPNSF)
Read Directory Entries (QHFRDDR)
Read from Stream File (QHFRDSF)
Rename Directory (QHFRNMDR)
Rename Stream File (QHFRNMSF)
Retrieve Directory Entry Attributes (QHFRTVAT)
Set Stream File Size (QHFSETSZ)
Write to Stream File (QHFWRTSF)

Change Directory Entry Attributes (QHFCHGAT) API

Parameters

Required Parameter Group:

1	Path name	Input	Char(*)
2	Path name length	Input	Binary(4)
3	Attribute information table	Input	Char(*)
4	Length of the attribute information table	Input	Binary(4)
5	Error code	I/O	Char(*)

The Change Directory Entry Attributes (QHFCHGAT) API changes the attributes of a specified directory entry for an existing file or directory.[1] As long as no other job has opened the directory in which the file or directory is located with a deny write lock, your application can add, change, or delete directory entry attributes.

Required Parameter Group

Path name
INPUT; CHAR(*)
The path name of the directory or file whose attributes you want to change. The directory or file must exist, and the path name must have more than one element. You cannot change the directory entry attributes of a file system.

Path name length
INPUT; BINARY(4)
The length of the path name, in bytes.

Attribute information table
INPUT; CHAR(*)
The table specifying the directory entry attributes to change. The file system determines which standard and extended attributes you can specify. For descriptions of the standard attributes, see "Directory Entry Attributes" on page 12-2. For the format of the table, see "Attribute Information Table" on page 12-4.

Note: In the DLS file system, the only standard attribute you can change is QFILATTR. If any other standard attributes are specified, they are ignored.

Some interchange document profile (IDP) attributes, such as the profile GCID (DIA.CA04C701), cannot be changed or deleted. If you specify these attributes, no error is returned, and the attributes are not changed or deleted. All IDP attribute names have a prefix of DIA (the letters DIA plus a period); for information about which ones can be changed, see the *IDP Reference*.

You cannot change the QNAME attribute. If it is specified in the attribute information table, it is ignored.

To change the value of an existing attribute, specify the attribute name and a valid value. To add an attribute, specify an attribute name not yet associated with the directory entry and a valid attribute value. To delete an attribute, specify an existing attribute name and a null attribute value. If the attribute to be deleted does not exist in the directory entry, it is ignored.

[1] Changing directory entry attributes in the DLS file system requires *CHANGE authority to the objects to which the attributes apply.

Length of the attribute information table
INPUT; BINARY(4)
The length of the attribute information table.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F21 E File name not valid.
CPF1F22 E File not found.
CPF1F26 E File in use.
CPF1F27 E Authority not sufficient to access file.
CPF1F28 E Damaged file.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F42 E Attribute information table not valid.
CPF1F43 E Attribute name not valid.
CPF1F44 E Attribute value is not valid.
CPF1F46 E Use of reserved attribute name not allowed.
CPF1F48 E Path name not valid.
CPF1F52 E Error code not valid.
CPF1F61 E No free space available on media.
CPF1F62 E Requested function failed.
CPF1F63 E Media is write protected.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F81 E API specific error occurred.
CPF1F82 E Function not supported.
CPF1F83 E File system name &1 not found.
CPF1F85 E Not authorized to file system &1.
CPF1F87 E Missing or damaged exit program &2.
CPF1F97 E File system &1 in use.

Change File Pointer (QHFCHGFP) API

Parameters

Required Parameter Group:

1	Open file handle	Input	Char(16)
2	Move information	Input	Char(6)
3	Distance to move	Input	Binary(4) Unsigned
4	New offset	Output	Binary(4) Unsigned
5	Error code	I/O	Char(*)

The Change File Pointer (QHFCHGFP) API moves the file pointer a specified number of bytes forward or backward. (The **file pointer** is the current read/write position in the file.) This file pointer is used by the Read from Stream File (QHFRDSF) and Write to Stream File (QHFWRTSF) APIs. You can also use the QHFCHGFP API to determine the size of a file by moving the pointer to the end of the file.

Required Parameter Group

Open file handle
INPUT; CHAR(16)
The handle returned when the file was opened with the Open Stream File (QHFOPNSF) API.

Move information
INPUT; CHAR(6)
Additional information specifying the action to take. The 6 characters of this parameter are:

1 The pointer's starting location. The pointer is moved the distance you specify from this place. For example, specifying 0 here and 5 in the distance to move parameter moves the pointer to a new position 5 bytes from the start of the file. Valid values for the starting location are:
- **0** The beginning of the file.
- **1** The pointer's current location.
- **2** The end of the file. If the distance to move is zero, the new offset parameter returns the file's size.

2-6 Reserved. These characters must be set to blanks.

Distance to move
INPUT; BINARY(4)
The distance to move the pointer from the starting location, in bytes. A negative value parameter moves the pointer backward in the file. A positive value moves it forward.

The file pointer can be set to any location that can be supported by a 4-byte unsigned value. An error is returned only if the application tries to move the pointer to a negative position or an offset larger than the maximum value that can be stored in a 4-byte unsigned binary number.

Setting the file pointer beyond the end of the file is allowed, but it does not change the file's size. To change the file's size, see "Set Stream File Size (QHFSETSZ) API" on page 14-21 or "Write to Stream File (QHFWRTSF) API" on page 14-22.

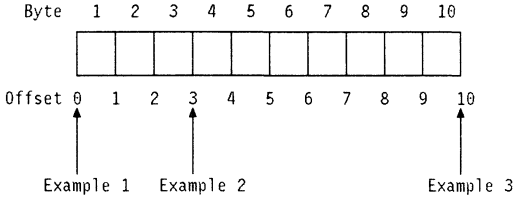
For brief examples of how the move information and the distance to move work together, see "How to Move the File Pointer" on page 14-3.

New offset
OUTPUT; BINARY(4) UNSIGNED
The new position of the pointer.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

How to Move the File Pointer

The file pointer represents a position or offset within a file where the next read or write is to take place. It does not actually point to a byte in the file; rather, it points to the gap between bytes. The diagram below represents a 10-byte file as a series of boxes. Each box represents a byte of data in the file.



Examples

1. To move the file pointer to the beginning of the file, set the start location in the move information parameter to zero (the beginning of the file). Set the distance to move to zero, too.
2. To move the file pointer to offset 3 (that is, pointing to the gap between bytes 3 and 4), set the start location in the move information parameter to zero (the beginning of the file). Set the distance to move to 3.
3. To move the file pointer to the end of the file, set the start location in the move information parameter to 2 (the end of the file). Set the distance to move to zero. The new offset parameter returns the file's size.

Error Messages

CPF1F2D	E	File pointer position not valid.
CPF1F2E	E	Range of bytes in file in use.
CPF1F25	E	File handle not valid.
CPF1F28	E	Damaged file.
CPF1F4E	E	Move information value not valid.
CPF1F4F	E	Distance to move value not valid.
CPF1F41	E	Severe error occurred while addressing parameter list.
CPF1F52	E	Error code not valid.
CPF1F62	E	Requested function failed.
CPF1F66	E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71	E	Exception specific to file system occurred.
CPF1F72	E	Internal file system error occurred.
CPF1F73	E	Not authorized to use command.
CPF1F74	E	Not authorized to object.
CPF1F81	E	API specific error occurred.
CPF1F82	E	Function not supported.
CPF1F87	E	Missing or damaged exit program &2.

Close Directory (QHFCLODR) API

Parameters

Required Parameter Group:

1	Open directory handle	Input	Char(16)
2	Error code	I/O	Char(*)

The Close Directory (QHFCLODR) API closes a specified directory that was opened using the Open Directory (QHFOPNDR) API. Once a directory is closed, the open directory handle that refers to it is no longer valid. If a process ends without closing directories, they are closed automatically. However, it is best if the application closes directories that it no longer needs so that other applications have free access to them.

Required Parameter Group

Open directory handle
INPUT; CHAR(16)
The directory handle returned by the QHFOPNDR API when the directory was opened.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F05	E	Directory handle not valid.
CPF1F08	E	Damaged directory.
CPF1F41	E	Severe error occurred while addressing parameter list.
CPF1F52	E	Error code not valid.
CPF1F62	E	Requested function failed.
CPF1F66	E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71	E	Exception specific to file system occurred.
CPF1F72	E	Internal file system error occurred.
CPF1F73	E	Not authorized to use command.
CPF1F74	E	Not authorized to object.
CPF1F81	E	API specific error occurred.
CPF1F82	E	Function not supported.
CPF1F87	E	Missing or damaged exit program &2.

Close Stream File (QHFCLOSF) API

Parameters

Required Parameter Group:

1	Open file handle	Input	Char(16)
2	Error code	I/O	Char(*)

The Close Stream File (QHFCLOSF) API closes the specified stream file, releasing any locks on the file or ranges within the file. Any data and directory information waiting to be written is forced to nonvolatile storage. **Nonvolatile**

storage is any storage area whose contents are not lost when power is cut off or when the system is loaded.

Required Parameter Group

Open file handle
INPUT; Input Char(16)
The handle returned when the file being closed was opened with the Open Stream File (QHFOPNSF) API.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F25 E File handle not valid.
CPF1F28 E Damaged file.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F52 E Error code not valid.
CPF1F61 E No free space available on media.
CPF1F62 E Requested function failed.
CPF1F63 E Media is write protected.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F81 E API specific error occurred.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.

Control File System (QHFCTLFS) API

Parameters

Required Parameter Group:

1	Open file handle	Input	Char(16)
2	File system name	Input	Char(10)
3	Input data buffer	Input	Char(*)
4	Input data buffer length	Input	Binary(4)
5	Output data buffer	Output	Char(*)
6	Output data buffer length	Input	Binary(4)
7	Length of data returned	Output	Binary(4)
8	Error code	I/O	Char(*)

The Control File System (QHFCTLFS) API transmits commands to your file system to be performed. The commands must be defined by the file system.

Required Parameter Group

Open file handle
INPUT; CHAR(16)
The identifier returned when the file was opened with the Open Stream File (QHFOPNSF) API. If you specify a file system in the file system name parameter, this parameter is ignored.

File system name
INPUT; CHAR(10)
The registered file system to send the request to (for example, QDLS). Valid values are:

name The name of the registered file system. The open file handle parameter is ignored.
***HANDLE** A special value indicating that the registered file system name is derived from the file handle provided in the open file handle parameter. Using the handle provides better performance than using the file system name.

Input data buffer
INPUT; CHAR(*)
The command string to send to the file system. This string differs from file system to file system. The QHFCTLFS API simply passes it on.

Input data buffer length
INPUT; BINARY(4)
The length of the input data buffer, in bytes.

Output data buffer
OUTPUT; CHAR(*)
The data returned from the file system.

Output data buffer length
INPUT; BINARY(4)
The length of the output data buffer, in bytes.

Length of data returned
OUTPUT; BINARY(4)
The length of the data returned by the file system in the output data buffer, in bytes.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F05 E Directory handle not valid.
CPF1F25 E File handle not valid.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F47 E Buffer overflow occurred.
CPF1F52 E Error code not valid.
CPF1F53 E Value for length of data buffer not valid.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F81 E API specific error occurred.

CPF1F82 E Function not supported.
CPF1F83 E File system name &1 not found.
CPF1F85 E Not authorized to file system &1.
CPF1F87 E Missing or damaged exit program &2.
CPF1F97 E File system &1 in use.

Copy Stream File (QHFCPYSF) API

Parameters

Required Parameter Group:

1	Source file path name	Input	Char(*)
2	Source file path name length	Input	Binary(4)
3	Copy information	Input	Char(6)
4	Target file path name	Input	Char(*)
5	Target file path name length	Input	Binary(4)
6	Error code	I/O	Char(*)

The Copy Stream File (QHFCPYSF) API copies an existing stream file into another stream file and optionally renames the copy.[2] The existing file being copied is called the **source file**. The copy, or the file that the source is copied into, is called the **target file**.

All file attributes except the revision date and time are copied from the source file to the target file. The file revision date and time are set to the current date and time. The file creation date and time stay as they are—that is, the source file's creation date and time.

Required Parameter Group

Source file path name
INPUT; CHAR(*)
The path name of the source file (the file being copied). The last element of the path name is the source file name.

The source file must be accessible. No other job can have the source file open with a deny read or deny read/write lock.

Source file path name length
INPUT; BINARY(4)
The length of the source file path name, in bytes.

Copy information
INPUT; CHAR(6)
The type of copy operation being performed. The 6 characters of this parameter are:

1 The action to take if the target file already exists. Valid values are:
- **0** Do not replace the existing file.
- **1** Replace the existing file with the copy.
- **2** Add the copy to the end of the existing file.

Note: In the DLS file system, you cannot add the copy to the existing file. If you specify this action, an error is returned.

2-6 Reserved. These characters must be blank.

Target file path name
INPUT; CHAR(*)
The path name of the target file (the copy or the file that the source is copied into). The last element of the path name is the target file name.

If the target file has a different name from the source file, it can be in the same path as the source.

The target file must be accessible in write mode. It cannot be a read-only file, and another job cannot have it open with a deny write or deny read/write lock.

Target file path name length
INPUT; BINARY(4)
The length of the target file path name, in bytes.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F2A E Number of open files exceeds limit.
CPF1F2E E Range of bytes in file in use.
CPF1F21 E File name not valid.
CPF1F22 E File not found.
CPF1F23 E New file name same as old file name.
CPF1F24 E File name already exists.
CPF1F26 E File in use.
CPF1F27 E Authority not sufficient to access file.
CPF1F28 E Damaged file.
CPF1F29 E Use of reserved file name not allowed.
CPF1F34 E Attempted write operation beyond file size limit.
CPF1F35 E Read file operation failed.
CPF1F36 E Write file operation failed.
CPF1F37 E File is a read-only file.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F42 E Attribute information table not valid.
CPF1F43 E Attribute name not valid.
CPF1F44 E Attribute value is not valid.
CPF1F46 E Use of reserved attribute name not allowed.
CPF1F48 E Path name not valid.
CPF1F51 E Copy information value not valid.
CPF1F52 E Error code not valid.

[2] Copying a DLS file requires *USE access to the source file. If you are creating a new file, you also need *CHANGE authority to the directory in which the new file is located. If the target file is being replaced, you need *CHANGE authority to that file.

CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F84 E	Operation across file systems not allowed.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.

Create Directory (QHFCRTDR) API

Parameters

Required Parameter Group:

1	Path name	Input	Char(*)
2	Length of path name	Input	Binary(4)
3	Attribute information table	Input	Char(*)
4	Length of attribute information table	Input	Binary(4)
5	Error code	I/O	Char(*)

The Create Directory (QHFCRTDR) API creates a new directory and its directory entry.[3] Except for the directory being created, all directories in the path must exist.

Required Parameter Group

Path name
INPUT; CHAR(*)
The path name for the new directory. The last element of the path name specifies the directory being created. For example, specifying /QDLS/A/B creates new directory B and adds a directory entry for B to directory A in file system QDLS.

Length of path name
INPUT; BINARY(4)
The length of the path name, in bytes.

Attribute information table
INPUT; CHAR(*)
The table specifying the attributes for the new directory. The file system determines which standard and extended attributes you can specify.[4] For detailed descriptions of the standard attributes and the format of the table, see "Directory Entry Attributes" on page 12-2.

If no attributes are specified, the file system's defaults are used for required information.

Length of attribute information table
INPUT; BINARY(4)
The length of the attribute information table, in bytes. Valid values are:

length	Use the attributes specified in the table.
0	Use the system defaults for standard attributes instead of using the attributes in the table.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F04 E	Directory name already exists.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F09 E	Use of reserved directory name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.

[3] Creating a directory entry in the DLS file system requires *CHANGE authority to the directory in which the new directory is being created.

[4] When you create a directory in the DLS file system, the system automatically sets the directory's creation date to the current date. You can specify only two attributes, text description and profile GCID (graphic code-page identifier), described in "Attributes to Use When Creating Directories and Files" on page 13-4. If you do not specify them, the file system uses its defaults. If you specify other DLS-defined attributes, an exception is returned. If you specify any standard attributes, they are ignored.

After creating a DLS directory, you can add other attributes with the Change Directory Entry Attributes (QHFCHGAT) API described on page 14-1.

CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.

Delete Directory (QHFDLTDR) API

Parameters

Required Parameter Group:

1	Path name	Input	Char(*)
2	Length of path name	Input	Binary(4)
3	Error code	I/O	Char(*)

The Delete Directory (QHFDLTDR) API deletes a single, empty directory. It cannot contain any directory entries. If this job or another job has already opened the directory with a deny none or deny write lock, it cannot be deleted.[5]

Required Parameter Group

Path name
INPUT; CHAR(*)
The path name of the directory being deleted. The last element of the path name must be a directory name. You cannot use a one-element path name specifying only the file system.

Path name length
INPUT; BINARY(4)
The length of the path name, in bytes.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F0A E	Delete directory operation not allowed.
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.

Delete Stream File (QHFDLTSF) API

Parameters

Required Parameter Group:

1	Path name	Input	Char(*)
2	Path name length	Input	Binary(4)
3	Error code	I/O	Char(*)

The Delete Stream File (QHFDLTSF) API deletes a single stream file.[6] Both the directory entry associated with the file and all data contained in the file object are deleted.

Required Parameter Group

Path name
INPUT; CHAR(*)
The path name for the file being deleted. The last element of the path name is the file name.

The file cannot be open or in use, and it cannot be a read-only file.

Path name length
INPUT; BINARY(4)
The length of the path name, in bytes.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.

[5] In the DLS file system, a directory opened with a lock mode of no lock cannot be deleted either. The DLS file system automatically changes a lock mode of no lock to deny none.

[6] Deleting a DLS file requires *ALL authority to that file.

CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.

Force Buffered Data (QHFFRCSF) API

Parameters

Required Parameter Group:

1	Files to force	Input	Char(16)
2	Error code	I/O	Char(*)

The Force Buffered Data (QHFFRCSF) API synchronously forces buffered data and directory entry information out of main storage and into nonvolatile storage for either a specific file or all files opened by a job. (**Nonvolatile storage** is any storage area whose contents are not lost when power is cut off or when the system is loaded.) Forcing buffered data is similar to closing a file because the data is forced. However, after a force operation, the file remains open, and forcing has no effect on locks and read/write positions.

Required Parameter Group

Files to force
INPUT; CHAR(16)
The files whose data is being forced. Valid values are:

Open file handle
Forces the single file that was assigned this handle when opened with the Open Stream File (QHFOPNSF) API.

Hexadecimal zeros
Forces all files opened by the job.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F2B E	Write operation not allowed to file opened for read only.
CPF1F2E E	Range of bytes in file in use.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F86 E	Force all files did not complete successfully.
CPF1F87 E	Missing or damaged exit program &2.

Get Stream File Size (QHFGETSZ) API

Parameters

Required Parameter Group:

1	Open file handle	Input	Char(16)
2	File size	Output	Binary(4) Unsigned
3	Error code	I/O	Char(*)

The Get Stream File Size (QHFGETSZ) API returns the current size of a stream file's data, in bytes, as of the last write operation to the file.

Required Parameter Group

Open file handle
INPUT; CHAR(16)
The file handle returned when the file was opened with the Open Stream File (QHFOPNSF) API.

File size
OUTPUT; BINARY(4) UNSIGNED
The number of bytes of data in the file as of the last write operation. This number is either the last offset written plus 1 or the size set with the Set Stream File Size (QHFSETSZ) API whichever is higher.

This size can differ from the value of the QFILSIZE attribute, which is the size of the file as of the last close operation. In addition, it is the size of the file's data only and does not include the size of any object

information stored by the file system. If your file system stores object or other information with the file, the file's allocated size is larger than this size.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F25 E File handle not valid.
CPF1F28 E Damaged file.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F52 E Error code not valid.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F81 E API specific error occurred.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.

List Registered File Systems (QHFLSTFS) API

Parameters

Required Parameter Group:

1	User space and library	Input	Char(*)
2	Format name	Input	Char(8)
3	Error code	I/O	Char(*)

The List Registered File Systems (QHFLSTFS) API resembles the Display Hierarchical File Systems (DSPHFS) command. The QHFLSTFS API retrieves a list of all file systems that are currently registered and thus available for use through the HFS APIs. The list gives the name, version level, and text description for each file system.

Required Parameter Group

User space and library
INPUT; CHAR(20)
The name of the *USRSPC object that is to receive the generated list. The first 10 characters give the user space name, and the second 10 characters give the name of the library in which the user space resides. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The format of the information returned. You must use this format name:

HFSL0100 File system list. For details, see "HFSL0100 Format."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

HFSL0100 Format

The HFSL0100 file system list consists of:

- A user area
- A generic header
- An input parameter section
- A list data section

The user area and generic header are described in "User Space Format for List APIs" on page 2-7. When you retrieve list entry information from a user space and you want to increase pointer values or add to variables used in the QUSRTVUS API, you must use the entry size returned in the generic header. The entries can be padded at the end.

The input parameter and list data sections are specific to the HFSL0100 format. They are described below. For a detailed description of each item, see "Field Descriptions."

Input Parameter Section

Offset			
Dec	**Hex**	**Type**	**Information**
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name

List Data Section

Offset			
Dec	**Hex**	**Type**	**Information**
0	0	CHAR(10)	File system name
10	A	CHAR(6)	Version
16	10	CHAR(50)	Text description

Field Descriptions

File system name. The name of the file system when it was registered.

Format name. The format of the returned output.

Text description. The description of the file system specified at registration time.

User space library name. The name of the library containing the user space.

User space name. The name of the user space that receives the list.

Version. The version the file system supports. This is in the form VxRxMx, where x represents the version, release, and modification levels, respectively.

Error Messages

CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F81 E	API specific error occurred.
CPF3C21 E	Format name &1 is not valid.
CPF8100 E	All CPF81xx messages could be signalled. xx is from 01 to FF.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9830 E	Cannot assign library &1.
CPF9838 E	User profile storage limit exceeded.

Lock and Unlock Range in Stream File (QHFLULSF) API

Parameters

Required Parameter Group:

1	Open file handle	Input	Char(16)
2	Lock information	Input	Char(6)
3	File offset where lock begins	Input	Binary(4) Unsigned
4	Bytes to lock	Input	Binary(4) Unsigned
5	File offset where unlock begins	Input	Binary(4) Unsigned
6	Bytes to unlock	Input	Binary(4) Unsigned
7	Error code	I/O	Char(*)

The Lock and Unlock Range in Stream File (QHFLULSF) API locks or unlocks a range of bytes in an open stream file. An application can lock part of a file instead of the entire file to temporarily keep other open instances from accessing that part.

An **open instance** is the state of a file having been opened and assigned an open file handle. The same job can open a file more than once. At each open operation, the file is assigned a unique open file handle, and the system treats the resulting state of being open as unique. Locks obtained during one open instance are honored by later open instances, even when the later open instances occur during the same job.

A locked range can be located anywhere in a file, and locking beyond the end of the file is allowed. Locks on a range are independent of the lock mode specified when the file is opened with the Open Stream File (QHFOPNSF) API.

Once a lock is obtained, the specified access is denied to all other requests to access that range in the file. The specified access is denied until the range is explicitly unlocked, the file is explicitly closed by the job, or the file is implicitly closed when the job ends. Closing a file releases all locks on the file.

If both unlocking and locking are specified on the request, the range is unlocked first. When unlocking is complete, the range is locked. This allows one open instance to keep other open instances from using a range that must be unlocked and relocked.

Your application should keep track of the ranges it locks. The QHFLULSF API does not track them, and HFS does not provide an API that lists locks.

Required Parameter Group

Open file handle
INPUT; CHAR(16)
The file handle returned when the file was opened with the QHFOPNSF API.

Lock information
INPUT; CHAR(6)
Additional information specifying the action to take. Valid values for the 6 characters in this parameter are:

1 The lock mode, indicating what access other open instances can have to this range of the file. Valid values are:
- **0** No lock. Use this when requesting an unlock operation only.
- **2** Deny write. Other open instances have read-only access to the locked range. The range can overlap or include other ranges locked in deny write mode, but it cannot overlap or include other ranges locked in deny read/write mode.
- **4** Deny read/write. This is an exclusive lock mode that denies other open instances all access to the locked range. The range cannot overlap or include any other locked range.

2-6 Reserved. These characters must be blank.

File offset where lock begins
INPUT; BINARY(4) UNSIGNED
The number of bytes from the beginning of the file where the range to lock begins.

Bytes to lock
INPUT; BINARY(4) UNSIGNED
The number of bytes to lock. If this is an unlock operation only, use zero for this parameter.

File offset where unlock begins
INPUT; BINARY(4) UNSIGNED
The number of bytes from the beginning of the file where the range to unlock begins.

Bytes to unlock
INPUT; BINARY(4) UNSIGNED
The number of bytes to unlock. If this is a lock operation only, use zero for this parameter.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F2E E Range of bytes in file in use.
CPF1F2F E Unlock range of bytes in file failed.
CPF1F25 E File handle not valid.
CPF1F28 E Damaged file.
CPF1F32 E Number of locks on file exceeds limit.
CPF1F4B E Value for number of bytes not valid.
CPF1F4C E Lock information value not valid.
CPF1F4D E File offset value not valid.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F52 E Error code not valid.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F81 E API specific error occurred.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.

Move Stream File (QHFMOVSF) API

Parameters

Required Parameter Group:

1	Source file path name	Input	Char(*)
2	Source file path name length	Input	Binary(4)
3	Target file path name	Input	Char(*)
4	Target file path name length	Input	Binary(4)
5	Error code	I/O	Char(*)

The Move Stream File (QHFMOVSF) API moves a single stream file from one directory to another and optionally changes the file's name.[7] The file's attributes are not changed.

The QHFMOVSF API only moves stream files to a different path. To rename files within a path, see "Rename Stream File (QHFRNMSF) API" on page 14-19.

Required Parameter Group

Source file path name
INPUT; CHAR(*)
The path name of the file being moved. The file name is the last element of the path name.

The source file must be accessible. You cannot move a file that is already in use by another job.

Source file path name length
INPUT; BINARY(4)
The length of the source file path name, in bytes.

Target file path name
INPUT; CHAR(*)
The path name designating the new location and, optionally, the new name of the file being moved.

The target file cannot already exist. It must reside in a different path from the source file. The file name can be the same as or different from the source file name.

Target file path name length
INPUT; BINARY(4)
The length of the target file path name, in bytes.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F03 E New directory name same as old directory name.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F2A E Number of open files exceeds limit.
CPF1F2E E Range of bytes in file in use.
CPF1F21 E File name not valid.
CPF1F22 E File not found.
CPF1F24 E File name already exists.
CPF1F26 E File in use.
CPF1F27 E Authority not sufficient to access file.
CPF1F28 E Damaged file.
CPF1F29 E Use of reserved file name not allowed.
CPF1F34 E Attempted write operation beyond file size limit.
CPF1F35 E Read file operation failed.
CPF1F36 E Write file operation failed.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F42 E Attribute information table not valid.
CPF1F43 E Attribute name not valid.
CPF1F44 E Attribute value is not valid.
CPF1F46 E Use of reserved attribute name not allowed.

[7] Moving a DLS file requires *ALL authority to the source file and *CHANGE authority to both the source and the target directories.

CPF1F48 E	Path name not valid.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F83 E	File system name &1 not found.
CPF1F84 E	Operation across file systems not allowed.
CPF1F85 E	Not authorized to file system &1.
CPF1F87 E	Missing or damaged exit program &2.
CPF1F97 E	File system &1 in use.

Open Directory (QHFOPNDR) API

Parameters

Required Parameter Group:

1	Directory handle	Output	Char(16)
2	Path name	Input	Char(*)
3	Path name length	Input	Binary(4)
4	Open information	Input	Char(6)
5	Attribute selection table	Input	Char(*)
6	Length of attribute selection table	Input	Binary(4)
7	Error code	I/O	Char(*)

The Open Directory (QHFOPNDR) API opens the specified directory so its directory entries can be read.[8] At open time, the directory pointer points to the first entry in the directory. As directory entries are read using the Read Directory Entries (QHFRDDR) API, the directory pointer advances so that the next entries will be read during future read operations.

Opening a directory can help streamline information retrieval and protect the directory. For example, you might open a directory to:

- Improve performance when obtaining information from more than one directory entry.
- Prevent the directory from being renamed or deleted. A deny none lock mode lets other processes read or change the contents of the directory, but keeps them from renaming or deleting the directory itself.
- Prevent the contents of the directory from being changed. A deny write lock mode keeps other processes from adding to or otherwise changing the open directory's contents. Directories and files cannot be created in the directory until it is closed.

Required Parameter Group

Directory handle
OUTPUT; CHAR(16)
An identifier made up of arbitrary characters returned by the API and used to identify the directory for subsequent operations, such as reading and closing.

Path name
INPUT; CHAR(*)
The path name for the directory being opened. The path and directory must exist.

For the directory name (the last element of the path name), you can use either a specific name or a generic name.

If the last element in the path is a specific name, that directory is opened and all directory entries in the directory are available for subsequent read operations.

If the last element in the path is a generic name, it identifies the directory entries to make available for subsequent read operations; the previous name in the path specifies the directory to open. Directory entries that are in the directory to open and that match the generic name are made available.

You can use these special matching characters in generic names:

- `*` An asterisk stands for zero or more characters. You can use it anywhere in a string.
- `?` A question mark at the end of a string represents zero or one character. A question mark embedded in a string represents one character.

For example, /QDLS/BUSY/DEPT* indicates all directories and files that have names beginning with DEPT and that are located in directory BUSY in the QDLS file system.

Path name length
INPUT; BINARY(4)
The length of the path name, in bytes.

Open information
INPUT; CHAR(6)
Information about the type and mode of the open operation. The characters and their meanings are:

1 The lock mode, indicating what other jobs can do to the directory.

Note: When you open a first-level file or directory (that is, a file or directory at the topmost level within the file system) in the DLS file system, no locks are applied to the directory. Valid values are:

0 No lock. Other jobs can read, change, rename, or delete the directory.

Note: The DLS file system does not support

[8] Opening a directory in the DLS file system requires *USE authority to the directory.

no-lock mode when a directory is opened. If you request this mode, it substitutes deny none.

1 Deny none. Other jobs can read or change directory entries, but they cannot rename or delete the directory.

2 Deny write. Other jobs can read the contents of the directory, but they cannot change, rename, or delete it.

Note: The DLS file system does not support deny write mode. If you specify this mode, an error is returned.

2 The type of open operation to perform. Valid values are:

0 Normal.

1 Permanent. The directory can be closed in only two ways, explicitly by the Close Directory (QHFCLODR) API, or implicitly, when the job ends. End request and reclaim resource operations do not close the directory.

3-6 Reserved. These characters must be blank.

Attribute selection table

INPUT; CHAR(*)

The table specifying which attributes are available when reading directory entries. The file system determines which standard and extended attributes you can specify. For detailed descriptions of the standard attributes, see "Directory Entry Attributes" on page 12-2. For the format of the table, see "Attribute Selection Table" on page 12-4.

This parameter lets you choose which *attributes* of the directory entries are available for reading when the directory is open. It does not determine which *directory entries* can be read. Use the path name parameter to select the directory entries you want to read.

Length of the attribute selection table

INPUT; BINARY(4)

The length of the table, in bytes, or a special value indicating which attributes are made available. Valid values are:

length The attribute selection table parameter contains the attributes the application wants to make available.

0 Only the standard attribute QNAME is available.

-1 All attributes are available.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F43 E Attribute name not valid.
CPF1F45 E Attribute selection table not valid.
CPF1F48 E Path name not valid.
CPF1F49 E Open information value not valid.
CPF1F52 E Error code not valid.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F81 E API specific error occurred.
CPF1F82 E Function not supported.
CPF1F83 E File system name &1 not found.
CPF1F85 E Not authorized to file system &1.
CPF1F87 E Missing or damaged exit program &2.
CPF1F97 E File system &1 in use.

Open Stream File (QHFOPNSF) API

Parameters

Required Parameter Group:

1	Open file handle	Output	Char(16)
2	Path name	Input	Char(*)
3	Path name length	Input	Binary(4)
4	Open information	Input	Char(10)
5	Attribute information table	Input	Char(*)
6	Length of attribute information table	Input	Binary(4)
7	Action taken	Output	Char(1)
8	Error code	I/O	Char(*)

The Open Stream File (QHFOPNSF) API opens and optionally creates a single stream file.[9] Applications can use the QHFOPNSF API to perform these tasks:

- Open an existing file.
- Open and replace an existing file. You cannot perform this operation on read-only files. (See page 12-3 for more information about read-only files.)
- Create a new file and open it.
- Return an error if the specified file exists.
- Return an error if the specified file does not exist.

[9] Opening a DLS file for read access requires *USE authority to the file. Opening a DLS file for write access requires *CHANGE authority to the file.

Do not use the QHFOPNSF API to change the directory entry attributes for an existing file. Instead, see "Change Directory Entry Attributes (QHFCHGAT) API" on page 14-1.

When the file is opened, the file pointer is set to the first byte of the file. Subsequent read/write operations move or increase the value of the file pointer. To move it explicitly, see "Change File Pointer (QHFCHGFP) API" on page 14-2.

Required Parameter Group

Open file handle
OUTPUT; CHAR(16)
An identifier made up of arbitrary characters assigned by the API and used to refer to the file in subsequent operations.

Path name
INPUT; CHAR(*)
The path name for the file. The last element of the path name is the file name.

Path name length
INPUT; BINARY(4)
The length of the path name, in bytes.

Open information
INPUT; CHAR(10)
Whether or not to open the file, and what the opened file's characteristics are. Each character of this parameter has a specific meaning. The characters and their meanings are:

1 The action to take if the file already exists. Valid values are:
   - **0** Do not open the file. Return an error.
   - **1** Open the file. When an existing file is opened and the file size is extended, the file system might not define the value of the new bytes.
   - **2** Replace the existing file. This is equivalent to deleting and re-creating the file. The directory entry information is replaced.

2 The action to take if the file does not exist. Valid values are:
   - **0** Return an error.
   - **1** Create the file.

3 The write-through flag for the file. Valid values are:
   - **0** Write operations to nonvolatile storage can be asynchronous. (**Nonvolatile storage** is any storage area whose contents are not lost when power is cut off or when the system is loaded.) An **asynchronous** write operation returns control to the application immediately, so it can continue the operation. The write operation occurs at a later, unspecified time.
   - **1** Write operations to nonvolatile storage must be synchronous. A **synchronous** write operation returns control to the operation only after the write operation completes.

4 Reserved. This field must be blank.

5 The file's lock mode. The **lock mode** defines what operations other jobs can perform on the file. Valid values are:
   - **1** Deny None
   - **2** Deny Write
   - **3** Deny Read
   - **4** Deny Read/Write (exclusive)

   For a detailed description of lock modes, see "Lock and Access Modes" on page 14-15.

6 The file's **access mode**, indicating the application's access rights to the file. Valid values are:
   - **0** Read Only
   - **1** Write Only
   - **2** Read/Write

   For a detailed description of access modes, see "Lock and Access Modes" on page 14-15.

7 The type of open operation to perform. Valid values are:
   - **0** Normal
   - **1** Permanent. The file can be closed in only two ways: explicitly with the QHFCLOSF API, described in "Close Stream File (QHFCLOSF) API" on page 14-3, or implicitly when the job ends. The End Request (ENDRQS) and Reclaim Resource (RCLRSC) commands do not close the file.

8-10 Reserved. These characters must be blank.

Attribute information table
INPUT; CHAR(*)
The table specifying the attributes of the directory entry for this file. The file system determines which standard and extended attributes you can specify. For detailed descriptions of the standard attributes and the format of the table, see "Attribute Information Table" on page 12-4.

Note: When you create a file in the DLS file system, the system automatically sets the file's creation date to the current date. When you create or replace a DLS file, you can specify only these attributes:

- The standard attributes QFILSIZE and QFILATTR, described in "Directory Entry Attributes" on page 12-2. If you specify other standard attributes, they are ignored.
- The DLS-defined attributes file type, text description, and profile GCID, described in "Attributes to Use When Creating Directories and Files" on page 13-4. If you specify other DLS-defined attributes, an error is returned.

When you replace a DLS file, the system replaces all attributes except the creation date (QCRTDTTM), which remains unchanged. If the attribute information table provides attributes, they are used for the file. Otherwise, the system uses its default attributes.

Use this parameter only when creating a new file or replacing an existing file. It is ignored when opening an existing file.

Length of the attribute information table
INPUT; BINARY(4)
The length of the attribute information table, in bytes, or a special value indicating which attributes to use. Valid values are:

length Use the attributes contained in the attribute information table.
0 Use the system defaults for standard attributes.

Action taken
OUTPUT; CHAR(1)
One of these values, indicating the action taken by the file system:

1 The file already existed and was not replaced.
2 The file did not exist and was created.
3 The file already existed and was replaced.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Lock and Access Modes

Lock and access modes determine which operations jobs can perform on files. The following sections describe lock and access modes in detail.

Lock Modes: The lock mode determines the type of access your job lets other jobs have to the file. For example, if other jobs can continue reading a file but cannot write to it without impeding your job, specify deny write. This lock mode lets other jobs read the file but keeps them from writing to it.

When you assign a lock mode, it applies only to that specific occurrence of the file being open. If the same job opens the file more than once, the lock mode does not apply to the other occurrences of the file being open.

A file can be opened multiple times by different jobs as long as the lock modes specified on the open operations are compatible.

Any locks placed on a file opened with the QHFOPNSF API are removed when the file is closed with the Close Stream File (QHFCLOSF) API or when the job ends.

The lock modes you can specify when opening a file are:

Deny Read/Write
Access to the file is exclusive for the current job. The current job can perform additional open operations on the file. However, no other job can open the file in any lock mode until the current job either closes it or ends.

Deny Write
Other jobs can read but cannot write to the file. In other words, no other job can open the file with write access; the file must be closed first, or the current job must end.

Deny Read
No other job can read the file until the current job either closes it or ends.

Deny None
Other jobs can read and write to the file. However, they cannot delete, rename, move, or change the attributes of the file until the current job either closes it or ends.

Access Modes: The access mode characteristic determines the type of access your job needs to the file. For example, if your job requires read/write access and another job has already opened the file with a lock mode of deny none, your open request succeeds. However, if the file is open with a lock mode of deny write, your job is denied access.

The following table shows the results of opening and then trying to reopen the same file using all combinations of access and lock modes:

1st Open Operation (by your job)		2nd, 3rd, 4th Open Operation (by other jobs)											
		Deny Read/ Write			Deny Write			Deny Read			Deny None		
Lock Mode	Access Mode	R/O	R/W	W/O	R/O	R/W	W/O	R/O	R/W	W/O	R/O	R/W	W/O
Deny Read/ Write	R/O	N	N	N	N	N	N	N	N	N	N	N	N
	R/W	N	N	N	N	N	N	N	N	N	N	N	N
	W/O	N	N	N	N	N	N	N	N	N	N	N	N
Deny Write	R/O	N	N	N	Y	N	N	N	N	N	Y	N	N
	R/W	N	N	N	N	N	N	N	N	N	Y	N	N
	W/O	N	N	N	N	N	N	Y	N	N	Y	N	N
Deny Read	R/O	N	N	N	N	N	Y	N	N	N	N	N	Y
	R/W	N	N	N	N	N	N	N	N	N	N	N	Y
	W/O	N	N	N	N	N	N	N	N	Y	N	N	Y
Deny None	R/O	N	N	N	Y	Y	Y	N	N	N	Y	Y	Y
	R/W	N	N	N	N	N	N	N	N	N	Y	Y	Y
	W/O	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y

Key:

Y Open is allowed
N Open is denied
R/W Read/write
R/O Read only
W/O Write only

Error Messages

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F2A E Number of open files exceeds limit.
CPF1F21 E File name not valid.
CPF1F22 E File not found.
CPF1F24 E File name already exists.
CPF1F26 E File in use.
CPF1F27 E Authority not sufficient to access file.
CPF1F28 E Damaged file.
CPF1F29 E Use of reserved file name not allowed.
CPF1F37 E File is a read-only file.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F42 E Attribute information table not valid.
CPF1F43 E Attribute name not valid.
CPF1F44 E Attribute value is not valid.
CPF1F46 E Use of reserved attribute name not allowed.
CPF1F48 E Path name not valid.
CPF1F49 E Open information value not valid.
CPF1F52 E Error code not valid.
CPF1F61 E No free space available on media.
CPF1F62 E Requested function failed.
CPF1F63 E Media is write protected.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F81 E API specific error occurred.
CPF1F82 E Function not supported.
CPF1F83 E File system name &1 not found.
CPF1F85 E Not authorized to file system &1.
CPF1F87 E Missing or damaged exit program &2.
CPF1F97 E File system &1 in use.

Read Directory Entries (QHFRDDR) API

Parameters

Required Parameter Group:

1	Open directory handle	Input	Char(16)
2	Data buffer	Output	Char(*)
3	Data buffer length	Input	Binary(4)
4	Number of directory entries to read	Input	Binary(4)
5	Number of directory entries read	Output	Binary(4)
6	Length of data returned	Input	Binary(4)
7	Error code	I/O	Char(*)

The Read Directory Entries (QHFRDDR) API reads one or more directory entries from a directory opened with the

Open Directory (QHFOPNDR) API.[10] The QHFOPNDR API's path name parameter determines which directory entries are read. The QHFOPNDR API's attribute selection table determines what information is returned for each directory entry. For details about the QHFOPNDR API, see "Open Directory (QHFOPNDR) API" on page 14-12.

You must open a directory before reading from it. The open directory handle returned by the QHFOPNDR API is needed as input to the QHFRDDR API.

The QHFRDDR API reads directory entries sequentially. Each time the QHFRDDR API is called, the directory pointer is advanced by the number of directory entries returned in the data buffer. Subsequent calls of the QHFRDDR API return additional directory entries, until there are no more to return.

Required Parameter Group

Open directory handle
INPUT; CHAR(16)
The directory handle obtained when the directory was opened with the QHFOPNDR API.

Data buffer
OUTPUT; CHAR(*)
The buffer to hold the directory entry information returned. For the format, see "Data Buffer."

Data buffer length
INPUT; BINARY(4)
The length of the data buffer described in "Data Buffer." The buffer must be large enough to hold the requested attributes for at least one directory entry. If it is too small, the read operation fails and no data is returned. However, the length of data returned parameter contains the total number of bytes the file system tried to return for the next directory entry. The application should increase the data buffer size to at least that number and try the request again.

Number of directory entries to read
INPUT; BINARY(4)
The number of directory entries to place in the data buffer.

Number of directory entries read
OUTPUT; BINARY(4)
The number of directory entries actually placed in the data buffer. The value of this field is 0 when there are no more directory entries to read.

Length of data returned
OUTPUT; BINARY(4)
If the read operation is successful, this field contains the total number of bytes returned in the data buffer. If the read operation is not successful because the data buffer is not large enough to hold at least one entry, this field contains the number of bytes required to hold the requested attributes for the next directory entry.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Data Buffer

The data buffer holds the information returned about each directory entry. What information is returned depends mostly on what attributes are selected in the attribute selection table when the directory is opened with the QHFOPNDR API. However, the QNAME attribute is returned with every directory entry even though it is never specified in the attribute selection table. Thus, the data buffer always contains at least one piece of information about each directory entry found, its name. If the directory entry exists but cannot be read because of damage or a lock by another process, two pieces of information are returned: the name, given in the QNAME attribute, and the error that occurred, given in the QERROR attribute. For details about these attributes, see "Directory Entry Attributes" on page 12-2.

The data buffer has three logical parts:

1. The first field specifies the number of directory entries returned.
2. The next fields give the offsets to the directory entries returned. There is one offset field for each directory entry.
3. Next are the attribute information tables for the directory entries returned. There is one attribute information table for each directory entry.

The following table shows the format of the data buffer. The offset fields are repeated until the offsets for all directory entries are listed; the attribute information table for each directory entry is repeated in the same way.

The format of the data buffer is:

Type	Field
BINARY(4)	Number of directory entries returned.
BINARY(4)	Offset to the first directory entry.
BINARY(4)	Offset to the next directory entry, if more than one exists. This field is repeated for each directory entry returned.
Directory entry	Attribute information table for the first directory entry.

[10] Reading directory entries for DLS file system objects requires *USE authority to the entries being read. Entries for which you do not have *USE authority are not returned.

Type	Field
Directory entry	Attribute information table for the next directory entry, if more than one exists. This field is repeated for each directory entry returned.
Note: Each directory entry in the table is represented by the standard attribute information table described in "Attribute Information Table" on page 12-4. Offsets within the directory entries are from the beginning of the directory entry, not from the beginning of the data buffer.	

Error Messages

CPF1F05 E Directory handle not valid.
CPF1F08 E Damaged directory.
CPF1F4A E Value for number of directory entries not valid.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F47 E Buffer overflow occurred.
CPF1F52 E Error code not valid.
CPF1F53 E Value for length of data buffer not valid.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F81 E API specific error occurred.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.

Read from Stream File (QHFRDSF) API

Parameters

Required Parameter Group:

1	Open file handle	Input	Char(16)
2	Data buffer	Output	Char(*)
3	Bytes to read	Input	Binary(4)
4	Bytes actually read	Output	Binary(4)
5	Error code	I/O	Char(*)

The Read from Stream File (QHFRDSF) API reads a specified number of bytes from a stream file opened with an access mode of read only or read/write.

The read operation starts at the current position of the **file pointer**, the location in the file where the next read or write operation occurs. When a file is opened with the Open Stream File (QHFOPNSF) API the file pointer is set to the first byte of the file. You can move the pointer explicitly with the QHFCHGFP API; see "Change File Pointer (QHFCHGFP) API" on page 14-2 for details. After the read operation, the file pointer value is increased by the number of bytes actually read.

Required Parameter Group

Open file handle
INPUT; CHAR(16)
The file handle returned when the file was opened with the QHFOPNSF API. Your application must have opened the file with an access mode of read only or read/write.

Data buffer
OUTPUT; CHAR(*)
The buffer that holds the data read from the file.

Bytes to read
INPUT; BINARY(4)
The number of bytes to read from the file, starting at the current file pointer position. The number must be less than or equal to the length of the data buffer.

Note: The DLS file system can read a maximum of 16 766 960 bytes on one call to the QHFRDSF API. If there is a deny read/write lock on the range of bytes specified, the function fails. Bytes beyond the end of the file cannot be read.

Bytes actually read
OUTPUT; BINARY(4)
The number of bytes actually read and returned in the data buffer.

The value of this parameter equals the value of the bytes-to-read input parameter unless an error occurs or the end of the file is reached. Reaching the end of the file is not an error. When the file pointer reaches the end of the file, the file system stores the bytes actually read in the data buffer and sets the actual number of bytes read, which in this case is less than the number of bytes to read. The application can then detect the end of the file by comparing the number of bytes actually read to the number requested.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F2C E Read operation not allowed to file opened for write only.
CPF1F2E E Range of bytes in file in use.
CPF1F25 E File handle not valid.
CPF1F28 E Damaged file.
CPF1F35 E Read file operation failed.
CPF1F4B E Value for number of bytes not valid.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F52 E Error code not valid.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F81 E API specific error occurred.

| CPF1F82 E Function not supported.
| CPF1F87 E Missing or damaged exit program &2.

Rename Directory (QHFRNMDR) API

Parameters

Required Parameter Group:

1	Path name	Input	Char(*)
2	Path name length	Input	Binary(4)
3	New directory name	Input	Char(*)
4	New directory name length	Input	Binary(4)
5	Error code	I/O	Char(*)

The Rename Directory (QHFRNMDR) API renames a single directory.[11] You cannot rename the directory if another job has already opened it with a lock mode of deny none or deny write. (When a job opens a directory with the Open Directory (QHFOPNDR) API, it specifies a lock mode. The **lock mode** specifies what other jobs are allowed to do to the directory while the first job has it open.)

Required Parameter Group

Path name
INPUT; CHAR(*)
The path name of the directory being renamed. The last element of the path name must be a directory name. You cannot use a one-element path name specifying only the file system.

Path name length
INPUT; BINARY(4)
The length of the path name, in bytes.

New directory name
INPUT; CHAR(*)
The new name for the directory. Specify only the directory name. Do not specify the path name.

New directory name length
INPUT; BINARY(4)
The length of the new directory name, in bytes.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

| CPF1F01 E Directory name not valid.
| CPF1F02 E Directory not found.
| CPF1F03 E New directory name same as old directory name.
| CPF1F04 E Directory name already exists.
| CPF1F06 E Directory in use.
| CPF1F07 E Authority not sufficient to access directory.
| CPF1F08 E Damaged directory.
| CPF1F09 E Use of reserved directory name not allowed.
| CPF1F41 E Severe error occurred while addressing parameter list.
| CPF1F48 E Path name not valid.
| CPF1F52 E Error code not valid.
| CPF1F61 E No free space available on media.
| CPF1F62 E Requested function failed.
| CPF1F63 E Media is write protected.
| CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
| CPF1F71 E Exception specific to file system occurred.
| CPF1F72 E Internal file system error occurred.
| CPF1F73 E Not authorized to use command.
| CPF1F74 E Not authorized to object.
| CPF1F75 E Error occurred during start-job-session function.
| CPF1F81 E API specific error occurred.
| CPF1F82 E Function not supported.
| CPF1F83 E File system name &1 not found.
| CPF1F85 E Not authorized to file system &1.
| CPF1F87 E Missing or damaged exit program &2.
| CPF1F97 E File system &1 in use.

Rename Stream File (QHFRNMSF) API

Parameters

Required Parameter Group:

1	Path name	Input	Char(*)
2	Path name length	Input	Binary(4)
3	New file name	Input	Char(*)
4	New file name length	Input	Binary(4)
5	Error code	I/O	Char(*)

The Rename Stream File (QHFRNMSF) API renames a stream file in the same path.[12]

Required Parameter Group

Path name
INPUT; CHAR(*)
The path name for the file being renamed. The last element of the path name is the name of the file.

The file cannot be open or in use.

[11] In the DLS file system, renaming a directory requires *ALL authority to the directory being changed and *CHANGE authority to the directory in which it resides. If another job has opened the directory, you cannot rename it, regardless of which lock mode the other job specified in its open operation.

[12] Renaming a DLS file requires *ALL authority to the file being renamed and *CHANGE authority to the directory in which it resides.

Path name length
INPUT; BINARY(4)
The length of the path name, in bytes.

New file name
INPUT; CHAR(*)
The new name for the file. Do not include the path.

The new name must be unique. A file with this name cannot already exist in the path given in the path name parameter.

New file name length
INPUT; BINARY(4)
The length of the new file name, in bytes.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F21 E File name not valid.
CPF1F22 E File not found.
CPF1F23 E New file name same as old file name.
CPF1F24 E File name already exists.
CPF1F26 E File in use.
CPF1F27 E Authority not sufficient to access file.
CPF1F28 E Damaged file.
CPF1F29 E Use of reserved file name not allowed.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F48 E Path name not valid.
CPF1F52 E Error code not valid.
CPF1F61 E No free space available on media.
CPF1F62 E Requested function failed.
CPF1F63 E Media is write protected.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F81 E API specific error occurred.
CPF1F82 E Function not supported.
CPF1F83 E File system name &1 not found.
CPF1F85 E Not authorized to file system &1.
CPF1F87 E Missing or damaged exit program &2.
CPF1F97 E File system &1 in use.

Retrieve Directory Entry Attributes (QHFRTVAT) API

Parameters

Required Parameter Group:

1	Path name	Input	Char(*)
2	Path name length	Input	Binary(4)
3	Attribute selection table	Input	Char(*)
4	Length of attribute selection table	Input	Binary(4)
5	Attribute information table	Output	Binary(4)
6	Length of attribute information table	Input	Binary(4)
7	Length of data returned	Output	Binary(4)
8	Error code	I/O	Char(*)

The Retrieve Directory Entry Attributes (QHFRTVAT) API retrieves attribute information from a specified directory entry for a directory or file.[13] The QHFRTVAT API might be faster and more efficient than explicitly opening, reading, and then closing the directory, even if your file system automatically opens and closes the directory during its retrieve operation.

You can use the QHFRTVAT API to determine whether a specific directory entry exists, as well as to get one or more attributes of a specific directory entry. The QHFRTVAT API works with only one directory entry at a time. To retrieve the attributes of several directory entries at once, see "Read Directory Entries (QHFRDDR) API" on page 14-16 and "Open Directory (QHFOPNDR) API" on page 14-12.

Required Parameter Group

Path name
INPUT; CHAR(*)
The path name of the directory or file to retrieve attributes from. The directory or file must exist, and the path name must have more than one element. You cannot retrieve directory entry attributes for a file system.

Note: In addition, in the DLS file system, you cannot retrieve the attributes of a file that has been opened by another job with a lock mode of deny read/write.

Path name length
INPUT; BINARY(4)
The length of the path name, in bytes.

Attribute selection table
INPUT; CHAR(*)
The table specifying the attributes to be returned in the attribute information table. The file system determines which standard and extended attributes you can specify. For descriptions of the standard attributes, see "Directory Entry Attributes" on page 12-2. For the

[13] Retrieving directory entry attributes for DLS file system objects requires *USE authority to the object to which the attributes apply.

format of the table, see "Attribute Selection Table" on page 12-4.

Note: In the DLS file system, you can specify only standard attributes and attributes stored in the interchange document profile (IDP). If you specify any other attribute, the data returned contains an element for that attribute, but the length of the attribute value is set to zero.

Length of the attribute selection table
INPUT; BINARY(4)
The length of the attribute selection table, in bytes, or a special value indicating which attributes are returned. Valid values are:

length The attribute selection table parameter contains the attributes the application wants to make available.
0 No attributes are returned. You can use this to see whether the directory entry exists.
-1 All attributes are returned.

Attribute information table
OUTPUT; CHAR(*)
The directory entry information returned, as specified in the attribute selection table parameter. For the format of the table containing the returned information, see "Attribute Information Table" on page 12-4.

Length of the attribute information table
INPUT; BINARY(4)
The length of the attribute information table. The table must be large enough to hold all the attributes requested. If it is too small, the retrieve operation fails and no attribute information is returned; however, the length of data returned parameter contains the number of bytes the file system tried to return for that directory entry. The application should increase the attribute information table's length to at least that size and try the request again.

Length of data returned
OUTPUT; BINARY(4)
If the retrieve operation is successful, this field contains the total number of bytes returned in the attribute information table.

If the retrieve operation fails because the attribute information table is too small to hold all of the attributes requested, this field contains the number of bytes required to hold the requested attributes.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F21 E File name not valid.
CPF1F22 E File not found.
CPF1F26 E File in use.
CPF1F27 E Authority not sufficient to access file.
CPF1F28 E Damaged file.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F42 E Attribute information table not valid.
CPF1F43 E Attribute name not valid.
CPF1F45 E Attribute selection table not valid.
CPF1F47 E Buffer overflow occurred.
CPF1F48 E Path name not valid.
CPF1F52 E Error code not valid.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F72 E Internal file system error occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F81 E API specific error occurred.
CPF1F82 E Function not supported.
CPF1F83 E File system name &1 not found.
CPF1F85 E Not authorized to file system &1.
CPF1F87 E Missing or damaged exit program &2.
CPF1F97 E File system &1 in use.

Set Stream File Size (QHFSETSZ) API

Parameters

Required Parameter Group:

1	Open file handle	Input	Char(16)
2	File size	Input	Binary(4) Unsigned
3	Error code	I/O	Char(*)

The Set Stream File Size (QHFSETSZ) API sets the size of a stream file in bytes. Applications can use the QHFSETSZ API to increase or decrease the size of a stream file that has been opened with write only or read/write access.

Existing locks on the file are maintained. If the entire file is locked, you cannot change its size. If part of the file is locked, the new size cannot interfere with the locked part. The description of the file size parameter in the following table discusses these restrictions.

Use these parameters to call the QHFSETSZ API in your application:

Required Parameter Group

Open file handle
INPUT; CHAR(16)
The handle returned when the file being enlarged or truncated was opened with the Open Stream File (QHFOPNSF) API. The file must have been opened with write only or read/write access.

File size
INPUT; BINARY(4) UNSIGNED
The size to make the file, in bytes. The size cannot start or end within a range of bytes locked in deny write or deny read/write mode, and it cannot extend beyond a locked range. If a locked range was not previously within the file, the file cannot be extended to include that range. If a locked range is within the file, the file cannot be truncated to exclude that range.

When the file size is increased, the underlying file system might not define the value of the new bytes. When the file size is decreased, the data in the truncated part of the file is lost.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F2B E	Write operation not allowed to file opened for read only.
CPF1F2E E	Range of bytes in file in use.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F4B E	Value for number of bytes not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F72 E	Internal file system error occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F81 E	API specific error occurred.
CPF1F82 E	Function not supported.
CPF1F87 E	Missing or damaged exit program &2.

Write to Stream File (QHFWRTSF) API

Parameters

Required Parameter Group:

1	Open file name	Input	Char(16)
2	Data buffer	Input	Char(*)
3	Bytes to write	Input	Char(*)
4	Bytes actually written	Output	Binary(4)
5	Error code	I/O	Char(*)

The Write to Stream File (QHFWRTSF) API writes bytes to a stream file. The file must have been opened with an access mode of write only or read/write. If there is a deny write or deny read/write lock on a byte range being written to, the function fails.

The write operation starts at the current position of the **file pointer**, the location in the file where the next read or write operation occurs. When a file is opened with the Open Stream File (QHFOPNSF) API the file pointer is set to the first byte of the file. You can move the pointer explicitly with the QHFCHGFP API; see "Change File Pointer (QHFCHGFP) API" on page 14-2 for details. After the write operation, the file pointer value is increased by the number of bytes written.

Required Parameter Group

Open file handle
INPUT; CHAR(16)
The file handle returned when the file is opened with the QHFOPNSF API.

Data buffer
INPUT; CHAR(*)
The buffer containing the data being written.

Bytes to write
INPUT; BINARY(4)
The number of bytes being written to the file, starting at the current file pointer position. The number must be less than or equal to the length of the data buffer.

Note: The DLS file system can write a maximum of 16 766 960 bytes on one call to the QHFWRTSF API. The maximum size of a DLS file is 2 147 483 647 bytes.

The application can write beyond the end of the file. This increases the file's size.

Bytes actually written
OUTPUT; BINARY(4)
The number of bytes actually written to the file. If an error occurs during the write operation, the value of this parameter can be less than the value of the bytes-to-write input parameter.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1F2B E	Write operation not allowed to file opened for read only.
CPF1F2E E	Range of bytes in file in use.
CPF1F25 E	File handle not valid.
CPF1F28 E	Damaged file.
CPF1F34 E	Attempted write operation beyond file size limit.
CPF1F36 E	Write file operation failed.
CPF1F4B E	Value for number of bytes not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F52 E	Error code not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.

| CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
| CPF1F71 E Exception specific to file system occurred.
| CPF1F72 E Internal file system error occurred.
| CPF1F73 E Not authorized to use command.
| CPF1F74 E Not authorized to object.
| CPF1F81 E API specific error occurred.
| CPF1F82 E Function not supported.
| CPF1F87 E Missing or damaged exit program &2.

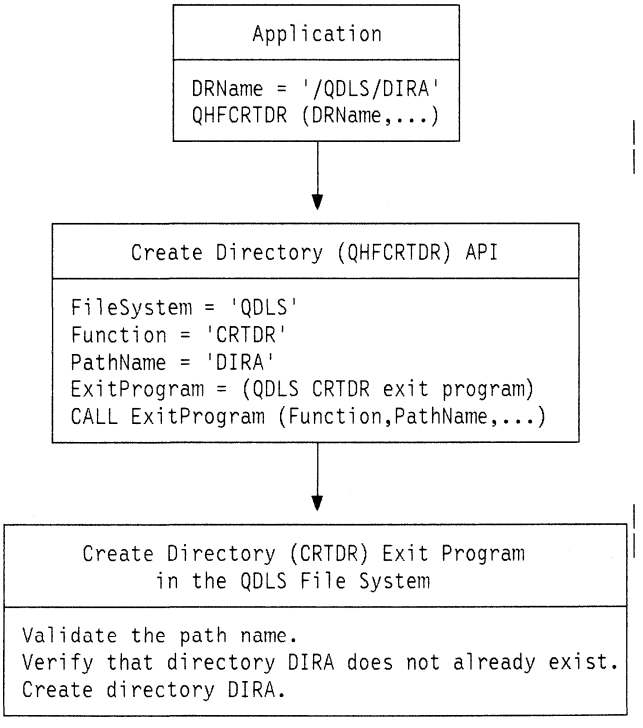
Chapter 15. Preparing to Use New File Systems with the HFS APIs

This chapter tells you how to create support for the HFS APIs so that application programmers can use them with new hierarchical file systems that you are creating or installing. If you simply want to use the HFS APIs to work with existing file systems that are already registered with HFS, see Part 5, "Hierarchical File System APIs."

To support the HFS APIs so that they can be used with your own file systems, you must supply two types of tools:

1. Exit programs that do the work for the HFS APIs. You must provide these exit programs. They enable application programs to use the specific HFS APIs to work with files and directories in your file system.

   Most exit programs correspond directly to APIs. For example, to allow applications to create directories in your file system, you must write an exit program that supplies the Create Directory function (CRTDR) for the Create Directory (QHFCRTDR) API. When an application calls an HFS API, the system routes the call to the appropriate exit program in the appropriate file system, according to the API's function and the file system name specified in the API's path-name parameter:



2. Programs that call the Register File System (QHFRGFS) and Deregister File System (QHFDRGFS) APIs. Registration makes your file system and its exit programs accessible to application programs using the HFS APIs. Deregistration lets you remove a file system from use.

The following sections give further instructions for preparing a new file system for use with HFS. Reference sections describing each API and exit program begin on page 15-6.

Enabling Your File System's Interface to HFS

Take these steps to make your file system available for use with the HFS APIs:

1. Read "Standard API and Exit Program Functions" on page 15-2 carefully. It describes which functions the HFS APIs perform for you and which functions your file system's exit programs must perform.
2. Define your own file and directory objects. These objects can be any of the following:
   - AS/400 objects, such as user spaces and user indexes
   - Objects on external devices attached to your AS/400 system
   - Objects on other systems that are attached to your AS/400 system by communications lines

   Your file system controls the structure, format, and location of the file and directory objects it defines. It also controls security and authority for those objects.
3. Create these programs:
   - A program or command to call the Register File System (QHFRGFS) API, which enrolls the file system for use with the HFS APIs.
   - A Start Job Session exit program, which is called the first time a job tries to use the file system. See "Start Job Session Exit Program" on page 15-6 for details.
   - An End Job Session exit program, which is called at job end to perform job cleanup. See "End Job Session Exit Program" on page 15-7 for details.
   - An exit program for every HFS API function that you want your file system to support. See the exit program and library list parameter of the Register File System (QHFRGFS) API on page 15-5 for a list of exit programs you can supply. If your file system is written in a high-level language that supports a variable number of input parameters, you can specify the same exit program for more than one function.

     Your file system does not need to support every function. However, if you supply an Open Stream File or Open Directory function to support the QHFOPNSF or QHFOPNDR API, you must supply the corresponding close function.
4. Register your file system with HFS, specifying the exit programs you want to make available. See "Register File System (QHFRGFS) API" on page 15-4 for complete instructions.

5. Give the file system's users authority to the Start Job Session exit program, described on page 15-6. Authority to this program gives users authority to the file system as a whole.
6. Give the file system's users authority to the HFS APIs that you support with exit programs and that you want to make public. A user needs *USE authority to an API to call the API from a program.
7. Provide the file system's users with complete documentation, so they know which HFS APIs are supported, and whether any of the HFS APIs work differently from the way they are described in this manual.

HFS Support and File System Job Processing

OS/400 HFS support and your file system work together to perform the function that an application requests when it calls an HFS API. **HFS support** is the part of the system that manages the HFS APIs as a group and passes information between the HFS APIs and the file system.

The first time an application or job specifies a particular file system in a call to an HFS API, HFS support does the following:

1. Checks the job's authority to the file system by checking its authority to the Start Job Session exit program. The job must have at least *USE authority to the Start Job Session exit program.
2. Obtains a shared read (*SHRRD) lock on the Start Job Session exit program. This prevents other jobs from deregistering the file system while the current job is using it. This lock remains in effect until the current job ends.
3. Calls the Start Job Session exit program, supplying the file system's name on the call in case the file system has been renamed during installation.

   The Start Job Session exit program performs any setup that the file system requires and returns a job handle to HFS support. The job handle is an arbitrary identifier that HFS support passes to the file system to help the file system keep track of this job. If the job calls another HFS API for this file system, HFS support passes this job handle as a parameter. The handle is treated as if it were a pointer, but it does not have to contain pointer data.
4. Calls the file system to complete the job's request by performing the work for the API.

On subsequent calls to the file system, HFS support retrieves the job handle for that file system and calls the file system exit program for the appropriate API. The file system can use the Start Job Session exit program only at the start of a job.

When OS/400 work management notifies HFS support that the job has ended, HFS support checks to see if the job has left any files or directories open. If it has, HFS support calls the file system to close them. HFS support then calls the End Job Session exit program to clean up any work areas used by the job. Your file system can use the End Job Session exit program to destroy temporary spaces and remove outstanding locks that were created during the job.

After the End Job Session exit program is run, HFS support unlocks the Start Job Session exit program, which was locked when the job first called the file system through an HFS API. The file system is no longer in use and can be deregistered.

For example, when the Start Job Session exit program is called, the file system could create a user space in the job's QTEMP library and return a space pointer to that user space as the job handle. On subsequent calls to HFS APIs for that file system, HFS support would pass that space pointer to the file system as the job handle. When the job ends, the file system's End Job Session exit program is called. The file system could use that exit program to delete the user space.

The exceptions that file systems are allowed to use are listed after each of the HFS exit programs. When a file system returns an allowed exception after a call from an HFS API, HFS support either sends the exception to the application or fills in the error code.

Standard API and Exit Program Functions

The HFS APIs and the file system's exit programs share the responsibility for validating input data, performing the functions requested by the application, reporting errors, and so on. The two sections that follow describe the standard functions that the HFS APIs perform for you and the standard functions that your exit programs must perform. A few APIs perform additional functions, and a few exit programs have additional requirements; these additions are listed at the end of each exit program description.

Standard API Functions

This section describes the functions that the HFS APIs and HFS support perform for your file system. When an application calls an HFS API, it appears to the application that the HFS API performs all the resulting functions. In reality, the API and HFS support perform only some of the functions. The file system exit program performs the rest.

Use this information to plan and create exit programs to support the HFS APIs in new file systems. You do not need this information to use the HFS APIs in your applications.

In general, every HFS API performs these functions for your file system:

- Automatically calls the file system's Start Job Session exit program the first time a job refers to the file system in a call to an HFS API. For a detailed explanation, see "Start Job Session Exit Program" on page 15-6.
- Processes the path name parameter as follows:

  - Extracts the file system name from the path name parameter.
  - Verifies that the file system is registered for use with the HFS APIs.
  - Passes the part of the path name that follows the file system name to the file system. For example, if the path name received from the application is /QDLS/A/B/C, then /A/B/C is passed to file system QDLS.

    There is one exception to this: The Open Directory (QHFOPNDR) API allows jobs to open the directory representing the file system itself. In that case, the path name consists of a slash and a single element, such as /QDLS, and the API passes just the slash (/) to the file system.
  - Recomputes the length of the path and directory name by subtracting the length of the extracted file system name, and passes the new length to the file system.
  - Verifies that there is at least one valid byte (the leading /) in the rest of the path and directory name parameter.
- Verifies that a directory or file handle passed from the application to the API is valid, and looks up the corresponding directory or file handle to pass from the API to the file system.
- Verifies that API parameters contain allowable values, and that reserved portions are set to blanks. API parameters include all fixed-length character fields like the open information parameter of the Open Stream File (QHFOPNSF) API.
- Verifies that length parameters contain allowable values. These length parameters are verified:
  - Length of attribute selection table
  - Length of attribute information table
  - Length of data buffer to read directory entries into
- Verifies that numeric or counting parameters contain the correct value. These parameters are verified:
  - Number of directory entries to read
  - Number of bytes to read
  - Number of bytes to write
- Calls the appropriate file system exit program to perform the operation.
- Monitors for valid exceptions from the file system, and either returns these to the application or maps them into an error code, as the application requests in the API's error code parameter.
- Signals successful completion of the operation to the application by not returning any errors.
- Returns control to the application.

Standard Exit Program Requirements

In general, every exit program you provide to support an HFS API must perform these functions:

- Verifies that the application has authority to perform the requested operation on the specified objects.
- Verifies that parameter values meet the file system's criteria.
- Verifies that directories and files named in the path name parameter either exist in the file system or, during operations like create and rename, that new names conform to the file system's naming conventions.

  The path name that the API passes to the file system does not include the file system's name but only the rest of the path name. If an application calls the Open Directory (QHFOPNDR) API and passes a path name specifying only the directory comprising the file system, such as /QDLS, the API passes only the slash (/) to the file system.
- Maintains API INPUT parameters as is, without changing their contents. If the file system must change these parameters, it must move them to another storage location. INPUT parameters must have the same value on entry to the file system exit program as on exit from the exit program. The file system can change OUTPUT parameters when the requested operation succeeds.
- Accepts attributes in the attribute selection and information tables used by HFS support, validates the data contained in the tables, and returns the appropriate data in the appropriate format.

  HFS support uses two common formats for passing attribute information between the application and the file system. These formats are described in "Attribute Selection Table" on page 12-4 and "Attribute Information Table" on page 12-4. The attribute selection table specifies which attributes the file system should return to the application. The file system must read the table, determine which attributes have been selected, and return those attributes to the application in the attribute information table.

  The file system must use the attribute information table when communicating with the application. It must accept and return attributes in that format. For its own use, however, the file system can store the attribute information in whatever format is most convenient.
- Performs the requested operation and returns any requested status information or data to the API.
- If the operation does not succeed, returns an exception describing the error to the API.

  The file system should return only the exceptions defined for the API because HFS support monitors only for those messages. If the file system sends any other message, an "Internal file system error" message is returned to the application.

If the file system needs to send its own messages, it can use the "File system specific error" message defined for each API. The file system's message ID is sent as insert data.

File System Registration APIs

This section describes APIs for registering and deregistering file systems:

Register File System (QHFRGFS) registers a file system with the HFS APIs so that application programmers can use the APIs to work with the file system.

Deregister File System (QHFDRGFS) reverses file system registration, preventing applications from using the file system through the HFS APIs.

Register File System (QHFRGFS) API

Parameters

Required Parameter Group:

1	File system name	Input	Char(10)
2	Version	Input	Char(6)
3	Registration information	Input	Char(6)
4	List of exit programs and libraries	Input	Char(20) Array
5	File system description	Input	Char(50)
6	Error code	I/O	Char(*)

The Register File System (QHFRGFS) API adds a new file system to OS/400 HFS support and records the APIs that it supports so they are accessible to user applications. You must register a file system before users can access it with the HFS APIs.

Authorities and Locks

Exit Program Authority
*USE or higher for all exit programs being registered.

File System Authority
For information about users' authority to use the file systems you register, see "User Authorizations and Locks for File System Functions" on page 15-5.

Required Parameter Group

File system name
INPUT; CHAR(*)
The name of the file system being registered. The name can be 1-10 characters long. The first character must be a capital letter other than Q (Q is reserved for IBM-supplied file systems). The remaining characters can be any combination of capital letters (A-Z) and numbers (0-9). The name cannot contain lowercase letters (a-z), special characters, or quotation marks.

The file system name determines where calls to HFS APIs are sent. For example, a call that contains a path name beginning with NEWS, such as /NEWS/DIRA/FILE1, is sent to the NEWS file system.

Note: IBM preregisters the document library services (DLS) file system, QDLS. You cannot register or deregister it yourself.

Version
INPUT; CHAR(6)

The version indicates the level of HFS with which the file system is compatible. Use the format VxRxMx where x stands for the version, release, and modification levels, respectively. The valid version is:

V2R1M0 Version 2 Release 1 Modification Level 0

Registration information
INPUT; CHAR(6)
Additional information describing the actions to take during registration.

The characters in this parameter are:

1 Whether to register a file system that is already registered. This character lets you reregister a changed file system without deregistering it first. Valid values are:
- **0** Do not reregister the file system.
- **1** Reregister the file system.

2 Which type of cross-file-system copy or move operation to perform. This character is called the **copy or move indicator**. HFS support checks its value only when an application specifies different source and target file systems in calls to the Copy Stream File (QHFCPYSF) or Move Stream File (QHFMOVSF) API.

This character has no effect on operations within the same file system. For cross-file-system operations, it tells the QHFCPYSF and QHFMOVSF APIs whether to call this file system's copy and move exit programs to see if they can perform the operations. If they cannot, the API tries the exit programs for the other file system involved, and then calls a series of other exit programs (such as those that open, read from, and write to stream files) to perform the operation. The last method is the least efficient.

For a detailed explanation of cross-file-system copy and move processing, see "Exit Program for Copy Stream File (QHFCPYSF) API" on page 15-11.

Valid values for this character are:
- **0** Do not call this file system's Copy Stream File or Move Stream File exit program when performing cross-file-system operations. The copy and move exit programs for this file system cannot communicate directly with any other file system, so the APIs should not waste time trying them.
- **1** Call this file system's Copy Stream File or Move Stream File exit program when performing cross-file-system operations. This file system's copy and move exit programs might be able to perform cross-file-system operations in some cases, so the APIs should

try them before trying the less efficient copy and move method.

Note: The DLS file system uses a value of 0 for this character. Its copy and move exit programs do not perform cross-file-system operations; instead, a series of other exit programs performs the operations.

3-6 Reserved. These characters must be blank.

List of exit programs and libraries
INPUT; CHAR(20) ARRAY
An array listing the exit programs that perform the work for the HFS APIs. (An **array** is a list of items in a specific sequence.) The first 10 characters of each array element contain the exit program name, and the second 10 characters of each array element contain the name of the library in which the exit program resides.

If the file system being registered does not support a particular API and thus there is no exit program, specify *NONE for the program name and blanks for the library name. If an application calls an API for which there is no exit program, the API issues a message stating that the file system does not support that operation.

If the file system is written in a language that supports a variable number of input parameters, you can specify the same exit program for more than one function.

The sequence of array elements indicates the operation or API supported by the exit program specified there. For example, the exit program you specify in position 3 is called when an application calls the QHFCRTDR API. The sequence is as follows:

1. Start Job Session Operation *(required)*
2. End Job Session Operation *(required)*
3. Create Directory (for the QHFCRTDR API)
4. Open Directory (for the QHFOPNDR API)
5. Read Directory Entries (for the QHFRDDR API)
6. Close Directory (for the QHFCLODR API; *required if an Open Directory exit program is specified*)
7. Retrieve Directory Entry Attributes (for the QHFRTVAT API)
8. Change Directory Entry Attributes (for the QHFCHGAT API)
9. Delete Directory (for the QHFDLTDR API)
10. Rename Directory (for the QHFRNMDR API)
11. Open Stream File (for the QHFOPNSF API)
12. Read from Stream File (for the QHFRDSF API)
13. Write to Stream File (for the QHFWRTSF API)
14. Lock and Unlock Range in Stream File (for the QHFLULSF API)
15. Change File Pointer (for the QHFCHGFP API)
16. Force Buffered Data (for the QHFFRCSF API)
17. Get Stream File Size (for the QHFGETSZ API)
18. Set Stream File Size (for the QHFSETSZ API)
19. Close Stream File (for the QHFCLOSF API; *required if an Open Stream File exit program is specified*)
20. Copy Stream File (for the QHFCPYSF API)
21. Delete Stream File (for the QHFDLTSF API)
22. Move Stream File (for the QHFMOVSF API)
23. Rename Stream File (for the QHFRNMSF API)
24. Control File System (for the QHFCTLFS API)

File system description
INPUT; CHAR(50)
A brief description of the file system. This description is returned when applications use the List Registered File Systems (QHFLSTFS) or the Display Hierarchical File Systems (DSPHFS) command.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Authorizations and Locks for File System Functions: The exit program that performs the Start Job Session operation controls user authority and locking for the file system as a whole.[1] The Start Job Session exit program is called the first time a job uses a given file system. At that time, the system checks the job's authority to the Start Job Session exit program. Authority for this program gives the user authority to use all other valid exit programs for the file system. OS/400 HFS support also obtains a shared read (*SHRRD) lock on the exit program and maintains it until the end of the job so that other jobs cannot deregister the file system while it is in use. For details about this exit program, see "Start Job Session Exit Program" on page 15-6.

HFS support maintains system pointers to the file system's exit programs. HFS support uses these pointers when your exit programs are called. If a system pointer to any exit program except Start Job Session becomes inaccurate, as it does when the program is recompiled, HFS support updates the pointer. If the system program cannot be resolved to the object, an error is returned. You can change and recompile any exit program except the Start Job Session exit program without having to reregister your file system. If you need to recompile the Start Job Session exit program, you must reregister the file system so that the pointer can be resolved.

Error Messages

CPF1F52 E Error code not valid.
CPF1F74 E Not authorized to object.
CPF1F81 E API specific error occurred.
CPF1F9A E Exit program list not valid.
CPF1F9B E Reregister or deregister file system failed.
CPF1F91 E File system name not valid.
CPF1F93 E File system &1 already registered.
CPF1F94 E Exit program &2 not found.

[1] Authority to the document library services (DLS) file system is controlled differently. See "User Enrollment" on page 13-2 for details.

| CPF1F95 E Required exit program not specified.
| CPF1F96 E Version level &2 not valid.
| CPF1F98 E Registration or deregistration cannot be done now.
| CPF1F99 E Register information value not valid.

Deregister File System (QHFDRGFS) API

Parameters

Required Parameter Group:

1	File system name	Input	Char(10)
2	Error code	I/O	Char(*)

The Deregister File System (QHFDRGFS) API removes a file system and its functions from HFS support so that applications can no longer work with the file system through the HFS APIs. You can use the QHFDRGFS API to keep users from working with a file system while you upgrade to a new release.

If there are open files or directories in the file system being deregistered, HFS support automatically closes them before deregistering the file system. See "End Job Session Exit Program" on page 15-7 for details.

Required Parameter Group

File system name
INPUT; CHAR(10);
The name of the file system being deregistered.

Note: You cannot deregister the document library services (DLS) file system, QDLS.

For deregistration to succeed, the file system cannot be in use. If a job that called the file system is not yet complete, the file system's Start Job Session exit program is still locked on behalf of that job, and the file system is still in use.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

| CPF1F52 E Error code not valid.
| CPF1F81 E API specific error occurred.
| CPF1F87 E Missing or damaged exit program &2.
| CPF1F9B E Reregister or deregister file system failed.
| CPF1F92 E File system &1 not registered.
| CPF1F97 E File system &1 in use.
| CPF1F98 E Registration or deregistration cannot be done now.

HFS Exit Programs

This section describes exit programs to support the HFS APIs. You must create exit programs to support these APIs. The exit programs are presented in alphabetical order by API name.

Start Job Session Exit Program

Parameters

Required Parameter Group:

1	Operation (INIT)	Input	Char(5)
2	File system job handle	Output	Char(16)
3	File system name	Input	Char(10)

Before applications can use the HFS APIs with your file system, you must supply a Start Job Session exit program for the file system.

The Start Job Session exit program controls access to the file system as a whole for each job in which that file system is used. The first time an application refers to a specific file system within a job by calling any HFS API, the HFS API performs these operations before performing its own function:

1. Checks the application's authority to the Start Job Session exit program. Authority to the Start Job Session exit program provides authority to all other exit programs that are registered for use with the file system.
2. Locks the Start Job Session exit program in shared read (*SHRRD) mode to keep the file system from being deregistered while in use.
3. Calls the Start Job Session exit program. The Start Job Session exit program sets up a 16-byte area called a job handle for the file system to use during the job, and returns the job handle to the HFS API. The file system can use the job handle to store a pointer to a separate work area or as a work area in itself.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

Operation (INIT)
INPUT; CHAR(5)
The abbreviation for the operation being performed (the letters INIT, for "initialize," followed by a blank).

File system job handle
OUTPUT; CHAR(16)
The work area or job handle for use by the file system. The file system returns the job handle to HFS on the Start Job Session. On all subsequent API calls, HFS returns the job handle to the file system.

The file system can keep whatever you choose in the job handle. For example, the job handle might contain a pointer giving the address of another work area or a control block used by the file system.

File system name
INPUT; CHAR(10)
The name of the file system received from the application in the call to the HFS API. This parameter specifies which name the file system should use when it issues exceptions.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F75 E Error occurred during start-job-session function.

End Job Session Exit Program

Parameters

Required Parameter Group:

1	Operation (TERM)	Input	Char(5)
2	File system job handle	Input	Char(16)

You must supply an End Job Session exit program for your file system. HFS support calls the End Job Session exit program whenever a job that uses the HFS APIs ends. The End Job Session exit program cleans up any work areas that the job used. If your Start Job Session exit program creates temporary work spaces, use the End Job Session exit program to delete them.

Required Parameter Group: HFS support passes this information to the End Job Session exit program:

Operation (TERM)
INPUT; CHAR(5)
The abbreviation for the operation being performed (TERM, meaning end).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F76 E Error occurred during end-job-session function.

Exit Program for Change Directory Entry Attributes (QHFCHGAT) API

Parameters

Required Parameter Group:

1	Operation (CHGAT)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Path name	Input	Char(*)
5	Path name length	Input	Binary(4)
6	Attribute information table	Input	Char(*)
7	Length of attribute information table	Input	Binary(4)

Before applications can use the Change Directory Entry Attributes (QHFCHGAT) API with your file system, you must:

1. Write an exit program that performs the change attribute operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Change Directory Entry Attributes (QHFCHGAT) API" on page 14-1.
2. Give the exit program's name when you register the file system with the Register File System API, QHFRGFS.

After that, when an application calls the QHFCHGAT API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (CHGAT)
INPUT; CHAR(5)
The abbreviation for the operation being performed (CHGAT).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Path name length
INPUT; BINARY(4)

Attribute information table
INPUT; CHAR(*)

Length of the attribute information table
INPUT; BINARY(4)

API Functions: The QHFCHGAT API performs the standard functions described on page 15-2. The API does not validate the attribute information table in any way. It checks only the length parameter to make sure it has a valid value.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Validates the attribute information table.
- Changes valid attributes in the directory entry.
- Ignores a request to delete an attribute that does not exist for the directory entry.
- Ignores a request to change an attribute that cannot be changed or that does not apply to the directory entry. For example, the exit program must ignore requests to change the QFILSIZE attribute in a directory entry for a directory object and must ignore requests to change directories to files, or vice versa.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F21 E	File name not valid.
CPF1F22 E	File not found.
CPF1F26 E	File in use.
CPF1F27 E	Authority not sufficient to access file.
CPF1F28 E	Damaged file.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program for Change File Pointer (QHFCHGFP) API

Parameters

Required Parameter Group:

1	Operation (CHGFP)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Open file handle	Input	Char(16)
4	Move information	Input	Char(6)
5	Distance to move	Input	Binary(4) Unsigned
6	New offset	Output	Binary(4) Unsigned

Before applications can use the Change File Pointer (QHFCHGFP) API with your file system, you must:

1. Write an exit program that performs the change file pointer operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Change File Pointer (QHFCHGFP) API" on page 14-2.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCHGFP API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

Operation (CHGFP)
INPUT; CHAR(5)
The abbreviation for the operation being performed (CHGFP).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

Open file handle
INPUT; CHAR(16)

Move information
INPUT; CHAR(6)

Distance to move
INPUT; BINARY(4) UNSIGNED

New offset
OUTPUT; BINARY(4) UNSIGNED

API Functions: The QHFCHGFP API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Checks for an attempt to set the file pointer to a negative position (that is, before the start of the file) or a position beyond the maximum value allowed in a 4-byte unsigned binary number, and signals an error if either occurs.
- Moves the file pointer the specified distance from the specified starting location, and records the file pointer's new offset value.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F2D E	File pointer position not valid.
CPF1F2E E	Range of bytes in file in use.
CPF1F28 E	Damaged file.
CPF1F4E E	Move information value not valid.
CPF1F4F E	Distance to move value not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program for Close Directory (QHFCLODR) API

Parameters

Required Parameter Group:

1	Operation (CLODR)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Open directory handle	Input	Char(16)

Before applications can use the Close Directory (QHFCLODR) API with your file system, you must:

1. Write an exit program that performs the close directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Close Directory (QHFCLODR) API" on page 14-3.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCLODR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (CLODR)
INPUT; CHAR(5)
The abbreviation for the operation being performed (CLODR).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameter is the same as the parameter for the API.

Open directory handle
INPUT; CHAR(16)

API Functions: In addition to the standard functions described on page 15-2, the QHFCLODR API performs these functions for your file system:

- Validates the open directory handle to ensure that the directory is open and the current user profile is the user that opened it.
- Passes the corresponding file system handle to the file system.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and one additional function. The exit program should close the directory, releasing the lock that the user obtained when the directory was opened, and invalidate the directory handle so that it cannot be used again.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F08 E	Damaged directory.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program for Close Stream File (QHFCLOSF) API

Parameters

Required Parameter Group:

1	Operation (CLOSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Open file handle	Input	Char(16)

Before applications can use the Close Stream File (QHFCLOSF) API with your file system, you must:

1. Write an exit program that performs the close stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Close Stream File (QHFCLOSF) API" on page 14-3.

2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCLOSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (CLOSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (CLOSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameter is the same as the parameter for the API.

Open file handle
INPUT; CHAR(16)

API Functions: The QHFCLOSF API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Releases any byte locks that the job has on the file.
- Closes the file and invalidates the file system job handle so that the handle cannot be used again.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F28 E	Damaged file.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program for Control File System (QHFCTLFS) API

Parameters

Required Parameter Group:

1	Operation (CTLFS)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Input data buffer	Input	Char(*)
3	File Handle	Input	Char(16)
5	Input data buffer length	Input	Binary(4)
6	Output data buffer	Output	Char(4)
7	Output data buffer length	Input	Binary(4)
8	Length of data returned	Output	Binary(4)

Before applications can use the Control File System (QHFCTLFS) API with your file system, you must:

1. Write an exit program that performs the control file system operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Control File System (QHFCTLFS) API" on page 14-4.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCTLFS API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

Operation (CTLFS)
INPUT: CHAR(10)
The abbreviation for the operation being performed (CTLFS).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API:

File Handle
INPUT; CHAR(16)

Input data buffer
INPUT; CHAR(*)

Input data buffer length
INPUT; BINARY(4)

Output data buffer
OUTPUT; CHAR(*)

Output data buffer length
INPUT; BINARY(4)

Length of data returned
OUTPUT; BINARY(4)

API Functions: The QHFCTLFS API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F47 E Buffer overflow occurred.
CPF1F53 E Value for length of data buffer not valid.
CPF1F62 E Requested function failer.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Copy Stream File (QHFCPYSF) API

Parameters

Required Parameter Group:

1	Operation (CPYSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Source file path name	Input	Char(*)
5	Source file path name length	Input	Binary(4)
6	Copy information	Input	Char(6)
7	Target file path name	Input	Char(*)
8	Target file path name length	Input	Binary(4)
9	File system names	Input	Char(20)

Before applications can use the Copy Stream File (QHFCPYSF) API with your file system, you must:

1. Write an exit program that performs the copy stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Copy Stream File (QHFCPYSF) API" on page 14-5.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API. In the registration-information parameter of the QHFRGFS API, indicate whether this exit program can be used for copy operations involving two different file systems.

After that, when an application calls the QHFCPYSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (CPYSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (CPYSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Source file path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Source file path name length
INPUT; BINARY(4)

Copy information
INPUT; CHAR(6)

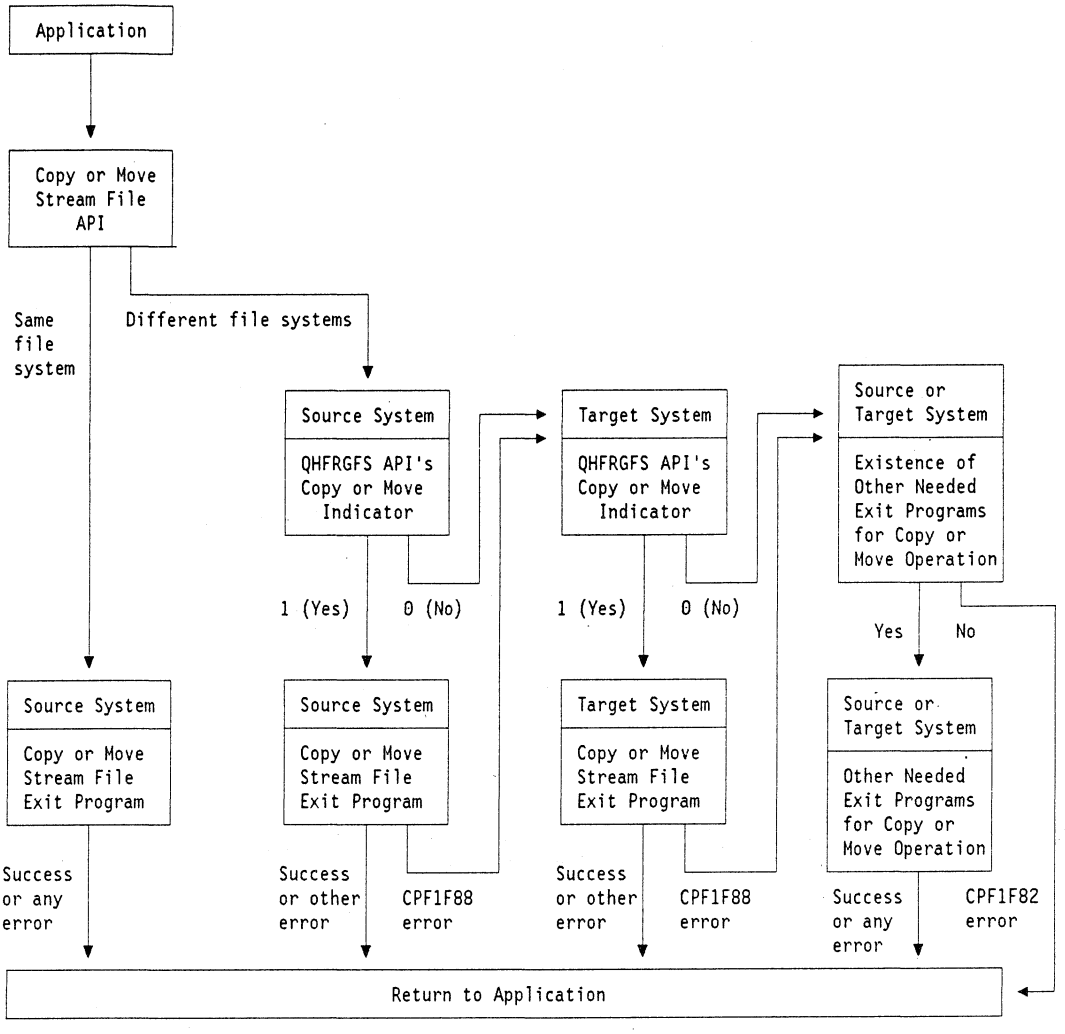
Target file path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Target file path name length
INPUT; BINARY(4)

File system names
INPUT; CHAR(20)
This is not an API parameter. The API derives this information from its source and target file path name parameters. The first 10 characters contain the name of the source file system, and the second 10 characters contain the name of the target file system.

API Functions: The QHFCPYSF API performs the standard functions described on page 15-2.

When the source and target file systems are different, the API performs additional functions so that the file is copied by the most efficient means available. The following diagram outlines the processing steps. The steps are the same for copy and move operations. They are described in detail after the diagram.



After determining that the file systems differ, the API tries to perform the copy operation in several different ways. If a method fails, the API proceeds to the next method. The possible methods are to call the following exit programs in the sequence listed:[2]

1. The source file system's Copy Stream File exit program
2. The target file system's Copy Stream File exit program
3. A series of other exit programs, such as those for opening, reading from, and writing to stream files, in the source and target file systems

The following paragraphs describe each method in detail.

1. **The source file system's Copy Stream File exit program:**

   First, the API checks the information provided when the source file system was registered with the Register File System (QHFRGFS) API. The copy or move indicator is character 2 of the registration-information parameter described on page 15-4.

   If the value of the source file system's copy or move indicator is 0 for no (indicating that this file system has no cross-file-system capability), the API proceeds to try the target file system.

   If the value of the target file system's copy or move indicator is 1 for yes (indicating that this file system has some cross-file-system capability), the API calls the file system's Copy Stream File exit program.

   If the exit program encounters an error in communicating with the target file system—for example, the exit program can work with some other file systems but not with this one—it returns message CPF1F88 to the API, which proceeds to the target file system.

   If the exit program completes the copy or encounters any other type of error—for example, the exit program cannot find the file to be copied—it returns control to the API. The API resends any errors received from the exit program to the application and then returns control to the application.

[2] The document library services (DLS) file system uses only the last method, the series of other exit programs.

2. **The target file system's Copy Stream File exit program:**

   The API follows the same procedure as for the source file system, checking the copy or move indicator and then trying the target file system's Copy Stream File exit program. If the copy or move indicator is 0 for no or if the exit program returns message CPF1F88, the API proceeds to try the other exit programs.

3. **A series of other exit programs in the source and target file systems:**

   If the source and target file systems' Copy Stream File exit programs have no cross-file-system capability at all, or if they have no such capability with respect to each other, the API might be able to perform a less efficient form of copy operation. If the following exit programs exist in the source and target file systems, the API tries to perform the copy operation. If any of these exit programs do not exist, the API returns message CPF1F82 and returns control to the application without performing the copy operation.

   The required exit programs in the source file system are:

   - Open Stream File (for the QHFOPNSF API)
   - Read from Stream File (for the QHFRDSF API)
   - Close Stream File (for the QHFCLOSF API)
   - Retrieve Stream File Attributes (for the QHFRTVAT API)
   - Delete Stream File (for the QHFDLTSF API). This exit program is required for both copy and move operations. However, it is used only during move operations.

   The required exit programs in the target file system are:

   - Open Stream File (for the QHFOPNSF API)
   - Write to Stream File (for the QHFWRTSF API)
   - Change File Pointer (for the QHFCHGFP API). This exit program is required for both copy and move operations. However, it is used only during copy operations in which the source file is being added to an existing target file.
   - Close Stream File (for the QHFCLOSF API)
   - Change Stream File Attributes (for the QHFCHGAT API)
   - Delete Stream File (for the QHFDLTSF API)

   The API uses the exit programs to perform their ordinary functions:

   - The source file system's Open Stream File exit program opens the source file, and the target file system's Open Stream File exit program opens the target file.
   - The source file system's Read from Stream File exit program and the target file system's Write to Stream File exit program repeatedly read from the source file and write to the target file until all the data has been copied from one to the other.
   - The source file system's Retrieve Stream File Attributes exit program retrieves the attributes stored with the source file, and the target file system's Change Stream File Attributes exit program writes the attributes to the target file.
   - The source and target file systems' Close Stream File exit programs close the source and target files.
   - The source file system's Delete Stream File exit program deletes the source file after a successful move operation.

     The target file system's Delete Stream File exit program is used during unsuccessful copy and move operations. If errors cause the failure of the operation as a whole, the exit program deletes any incomplete target files created during the operation. Target files that existed before the operation began are not deleted but might be partly changed.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- If the source and target paths to the files are the same, verifies that the source and target file names are different.
- If the source and target file systems are different, takes these actions:
  - Determines whether it is the source or the target file system by examining the exit program's file-system-names parameter.
  - Performs the copy operation or returns control to the application or API as described in "API Functions" on page 15-11.

    When the exit program has some cross-file-system capability but cannot complete this specific copy operation, it should return message CPF1F88 to the API. This tells the API that it might still be possible to perform the copy operation by other means. If unavoidable errors occur—for example, if the file being copied does not exist—then the exit program should return those errors to the API.

  Whether or not the QHFCPYSF API calls this exit program for cross-file-system operations depends on information provided when this file system was registered. If the second character of the registration-information parameter of the Register File System (QHFRGFS) API has a value of 1 (yes), the QHFCPYSF API calls this exit program. If that character has a value of 0 (no), the QHFCPYSF API does not call this exit program for cross-file-system operations; instead, the API bypasses this exit program and proceeds to try the next available method of copying files.
- If the operation is replacing or adding a copy to a file, verifies that the target file exists.

- Ensures that the user has the required authority to both the source and target paths and files.
- Ensures that the user has read access to the source file and write access to the target file.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F21 E File name not valid.
CPF1F22 E File not found.
CPF1F23 E New file name same as old file name.
CPF1F24 E File name already exists.
CPF1F26 E File in use.
CPF1F27 E Authority not sufficient to access file.
CPF1F28 E Damaged file.
CPF1F29 E Use of reserved file name not allowed.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F48 E Path name not valid.
CPF1F51 E Copy information value not valid.
CPF1F61 E No free space available on media.
CPF1F62 E Requested function failed.
CPF1F63 E Media is write protected.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F77 E Severe parameter error occurred on call to file system.
CPF1F88 E Unable to complete copy or move operation.

Note: You can use message CPF1F88 only when trying to perform cross-file-system copy operations. If you use it when copying files within a single file system, an `Internal file system error` message is returned to the application. You can use the exit program's file-system-names parameter to determine whether the move is across file systems.

Because this message does not always indicate an error, the application calling the QHFCPYSF API does not receive it. It is used only to communicate between the file system's exit program and the API.

Exit Program for Create Directory (QHFCRTDR) API

Parameters

Required Parameter Group:

1	Operation (CRTDR)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Path name	Input	Char(*)
5	Length of path name	Input	Char(*)
6	Attribute information table	Input	Char(*)
7	Length of attribute information table	Input	Binary(4)

Before applications can use the Create Directory (QHFCRTDR) API with your file system, you must:

1. Write an exit program that performs the create directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Create Directory (QHFCRTDR) API" on page 14-6.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFCRTDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (CRTDR)
INPUT; CHAR(5)
The abbreviation for the operation being performed (CRTDR).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Length of path name
INPUT; BINARY(4)

Attribute information table
INPUT; CHAR(*)

Length of attribute information table
INPUT; BINARY(4)

API Functions: The QHFCRTDR API performs the standard functions described on page 15-2 and one additional function. The API verifies that the length of the attribute information table is not negative. It does not validate the attribute information in any way.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Supports user-defined attribute types.
- Validates the attribute information.
- Associates the specified attributes with the directory when it is created.
- Returns an exception without creating a directory if another user has the higher-level directory locked in deny write mode. (The **higher-level directory** is the directory in which the new directory is being created.)

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F04 E	Directory name already exists.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F09 E	Use of reserved directory name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F42 E	Attribute information table not valid.
CPF1F43 E	Attribute name not valid.
CPF1F44 E	Attribute value is not valid.
CPF1F46 E	Use of reserved attribute name not allowed.
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program for Delete Directory (QHFDLTDR) API

Parameters

Required Parameter Group:

1	Operation (DLTDR)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Path name	Input	Char(*)
5	Path name length	Input	Binary(4)

Before applications can use the Delete Directory (QHFDLTDR) API with your file system, you must:

1. Write an exit program that performs the delete directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Delete Directory (QHFDLTDR) API" on page 14-7.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFDLTDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (DLTDR)
INPUT; CHAR(5)
The abbreviation for the operation being performed (DLTDR).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Path name length
INPUT; BINARY(4)

API Functions: The QHFDLTDR API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Verifies that the directory being deleted is not in use.
- Deletes the directory and its associated directory entry.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F0A E	Delete directory operation not allowed.
CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.

| CPF1F08 E Damaged directory.
| CPF1F41 E Severe error occurred while addressing parameter list.
| CPF1F48 E Path name not valid.
| CPF1F61 E No free space available on media.
| CPF1F62 E Requested function failed.
| CPF1F63 E Media is write protected.
| CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
| CPF1F71 E Exception specific to file system occurred.
| CPF1F73 E Not authorized to use command.
| CPF1F74 E Not authorized to object.
| CPF1F75 E Error occurred during start-job-session function.
| CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Delete Stream File (QHFDLTSF) API

Parameters

Required Parameter Group:

1	Operation (DLTSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Path name	Input	Char(*)
5	Path name length	Input	Binary(4)

Before applications can use the Delete Stream File (QHFDLTSF) API with your file system, you must:

1. Write an exit program that performs the delete stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Delete Stream File (QHFDLTSF) API" on page 14-7.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFDLTSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (DLTSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (DLTSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Path name length
INPUT; BINARY(4)

API Functions: The QHFDLTSF API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Ensures that the file being deleted is not open or in use.
- Ensures that the file being deleted is not a read-only file.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

| CPF1F01 E Directory name not valid.
| CPF1F02 E Directory not found.
| CPF1F06 E Directory in use.
| CPF1F07 E Authority not sufficient to access directory.
| CPF1F08 E Damaged directory.
| CPF1F21 E File name not valid.
| CPF1F22 E File not found.
| CPF1F26 E File in use.
| CPF1F27 E Authority not sufficient to access file.
| CPF1F28 E Damaged file.
| CPF1F41 E Severe error occurred while addressing parameter list.
| CPF1F48 E Path name not valid.
| CPF1F61 E No free space available on media.
| CPF1F62 E Requested function failed.
| CPF1F63 E Media is write protected.
| CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
| CPF1F71 E Exception specific to file system occurred.
| CPF1F73 E Not authorized to use command.
| CPF1F74 E Not authorized to object.
| CPF1F75 E Error occurred during start-job-session function.
| CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Force Buffered Data (QHFFRCSF) API

Parameters

Required Parameter Group:

1	Operation (FRCSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Files to force	Input	Char(16)

Before applications can use the Force Buffered Data (QHFFRCSF) API with your file system, you must:

1. Write an exit program that performs the force operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Force Buffered Data (QHFFRCSF) API" on page 14-8.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFFRCSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (FRCSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (FRCSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameter is the same as the parameter for the API.

Files to force
INPUT; CHAR(16)

API Functions: The QHFFRCSF API performs the standard functions described on page 15-2 and these additional functions:

- It verifies that the files to force parameter gives either a valid open file handle or hexadecimal zeros to indicate all files.
- If one file is specified, it calls the appropriate file system to force the data. Any exceptions received from the file system are either resignaled or mapped into an error code, as the application requests.
- If all files are specified, it calls each file system used by the process once for each open file within that file system to force the data for that file. Any exceptions received from the file system are left in the job log, and the API continues processing the remaining open files until all are forced. Any exceptions are reported to the caller as a special error condition indicating that the attempt to force all files partially failed.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F2B E Write operation not allowed to file opened for read only.
CPF1F2E E Range of bytes in file in use.
CPF1F28 E Damaged file.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F61 E No free space available on media.
CPF1F62 E Requested function failed.
CPF1F63 E Media is write protected.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Get Stream File Size (QHFGETSZ) API

Parameters

Required Parameter Group:

1	Operation (GETSZ)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Open file handle	Input	Char(16)
4	File size	Output	Binary(4) Unsigned

Before applications can use the Get Stream File Size (QHFGETSZ) API with your file system, you must:

1. Write an exit program that performs the get size operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Get Stream File Size (QHFGETSZ) API" on page 14-8.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFGETSZ API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

Operation (GETSZ)
INPUT; CHAR(5)
The abbreviation for the operation being performed (GETSZ).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

Open file handle
INPUT; CHAR(16)

File size
OUTPUT; BINARY(4) UNSIGNED

API Functions: The QHFGETSZ API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and one additional function. The exit program should retrieve the size of the file as of the last write operation, excluding any object information stored with the file.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F28 E Damaged file.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Lock and Unlock Range in Stream File (QHFLULSF) API

Parameters

Required Parameter Group:

1	Operation (LULSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Open file handle	Input	Char(16)
4	Lock information	Input	Char(6)
5	File offset where lock begins	Input	Binary(4)
6	Bytes to lock	Input	Binary(4) Unsigned
7	File offset where unlock begins	Input	Binary(4) Unsigned
8	Bytes to unlock	Input	Binary(4) Unsigned

Before applications can use the Lock and Unlock Range in Stream File (QHFLULSF) API with your file system, you must:

1. Write an exit program that performs the lock and unlock operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Lock and Unlock Range in Stream File (QHFLULSF) API" on page 14-10.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFLULSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (LULSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (LULSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

Open file handle
INPUT; CHAR(16)

Lock information
INPUT; CHAR(6)

File offset where lock begins
INPUT; BINARY(4) UNSIGNED

Bytes to lock
INPUT; BINARY(4) UNSIGNED

File offset where unlock begins
INPUT; BINARY(4) UNSIGNED

Bytes to unlock
INPUT; BINARY(4) UNSIGNED

API Functions: The QHFLULSF API performs the standard functions described on page 15-2 and these additional functions:

- Verifies that the bytes to lock and bytes to unlock parameters are not both zero. In other words, the API ensures that a lock, unlock, or both operations are to be performed. If both parameters are zero, an error is returned.
- Verifies that the lock information parameter contains valid values, and that the lock mode value is applicable to the lock or unlock operation requested.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Verifies that the file offsets to lock and unlock are valid offsets.
- Verifies that the number of bytes to lock and unlock are valid values.
- For an unlock operation, verifies that this job previously locked the range.

- For a lock operation, verifies that no part of the range already has a lock that does not allow the access requested.
- Unlocks or locks the range requested.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F2E E Range of bytes in file in use.
CPF1F2F E Unlock range of bytes in file failed.
CPF1F28 E Damaged file.
CPF1F32 E Number of locks on file exceeds limit.
CPF1F4B E Value for number of bytes not valid.
CPF1F4C E Lock information value not valid.
CPF1F4D E File offset value not valid.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Move Stream File (QHFMOVSF) API

Parameters

Required Parameter Group:

1	Operation (MOVSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Source file path name	Input	Char(*)
5	Source file path name length	Input	Binary(4)
6	Target file path name	Input	Char(*)
7	Target file path name length	Input	Binary(4)
8	File system names	Input	Char(20)

Before applications can use the Move Stream File (QHFMOVSF) API with your file system, you must:

1. Write an exit program that performs the move stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Move Stream File (QHFMOVSF) API" on page 14-11.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API. In the registration-information parameter of the QHFRGFS API, indicate whether this exit program can be used for move operations involving two different file systems.

After that, when an application calls the QHFMOVSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (MOVSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (MOVSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Source file path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Source file path name length
INPUT; BINARY(4)

Target file path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Target file path name length
INPUT; BINARY(4)

File system names
INPUT; CHAR(20)
This is not an API parameter. The API derives this information from its source and target file path name parameters. The first 10 characters contain the name of the source file system, and the second 10 characters contain the name of the target file system.

API Functions: The QHFMOVSF API performs the standard functions described on page 15-2.

When the source and target file systems are different, the API performs additional functions so that the file is moved by the most efficient means available. The processing steps are the same as those described for the Copy Stream File exit program in "API Functions" on page 15-11, with these exceptions:

- The Move Stream File (QHFMOVSF) API and Move Stream File exit program are used instead of the Copy Stream File API and exit program.
- After successful completion of the move operation, the source file system's Delete Stream File exit program is used to delete the source file.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- When the source and target file systems are different, performs the same actions described for the Copy Stream File exit program; see "Exit Program Requirements" on page 15-13.
- Verifies that the target file does not exist.
- Ensures that the file being moved is not open or in use.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F03 E New directory name same as old directory name.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F21 E File name not valid.
CPF1F22 E File not found.
CPF1F24 E File name already exists.
CPF1F26 E File in use.
CPF1F27 E Authority not sufficient to access file.
CPF1F28 E Damaged file.
CPF1F29 E Use of reserved file name not allowed.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F48 E Path name not valid.
CPF1F61 E No free space available on media.
CPF1F62 E Requested function failed.
CPF1F63 E Media is write protected.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F77 E Severe parameter error occurred on call to file system.
CPF1F88 E Unable to complete copy or move operation.

Note: You can use message CPF1F88 only when trying to perform cross-file-system move operations. If you use it when moving files within a single file system, an `Internal file system error` message is returned to the application. You can use the exit program's file-system-names parameter to determine whether the move is across file systems.

Because this message does not always indicate an error, the application calling the QHFMOVSF API does not receive it. It is used only to communicate between the file system's exit program and the API.

Exit Program for Open Directory (QHFOPNDR) API

Parameters

Required Parameter Group:

1	Operation (OPNDR)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Open directory handle	Output	Char(16)
5	Path name	Input	Char(*)
6	Path name length	Input	Binary(4)
7	Open information	Input	Char(6)
8	Attribute selection table	Input	Char(4)
9	Length of attribute selection table	Input	Binary(4)

Before applications can use the Open Directory (QHFOPNDR) API with your file system, you must:

1. Write an exit program that performs the open directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Open Directory (QHFOPNDR) API" on page 14-12.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFOPNDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

Operation (OPNDR)
INPUT; CHAR(5)
The abbreviation for the operation being performed (OPNDR).

File system job handle
INPUT; CHAR(16)
The work area or job identifier used by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Open directory handle
OUTPUT; CHAR(16)

Path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Path name length
INPUT; BINARY(4)

Open information
INPUT; CHAR(6)
The exit program can ignore character 2, which describes the type of open operation to perform. This field is used by HFS support during job cleanup if the job ends before the file is closed.

Attribute selection table
INPUT; CHAR(*)

Length of the attribute selection table
INPUT; BINARY(4)

API Functions: The QHFOPNDR API performs the standard functions described on page 15-2. The API does not validate the attribute selection table in any way.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Verifies that all directories in the path exist.
- If the last element of the path name is a specific name, opens that specific directory.
- If the last element of the path name is a generic name:
  - Opens the directory specified as the next-to-last element.
  - Interprets the generic name so that only directory entries that match it are available for subsequent read operations.
- Validates the attribute selection table.
- Locks the directory and its attributes according to the specified lock mode.
- Associates the attribute selection table with the open directory so that subsequent read operations using the QHFRDDR API return only the attributes selected when the directory was opened.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F43 E	Attribute name not valid.
CPF1F45 E	Attribute selection table not valid.
CPF1F48 E	Path name not valid.
CPF1F49 E	Open information value not valid.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program for Open Stream File (QHFOPNSF) API

Parameters

Required Parameter Group:

1	Operation (OPNSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Open file handle	Output	Char(16)
5	Path name	Input	Char(*)
6	Path name length	Input	Binary(4)
7	Open information	Input	Char(10)
8	Attribute information table	Input	Char(*)
9	Length of attribute information table	Input	Binary(4)
10	Action taken	Output	Char(1)

Before applications can use the Open Stream File (QHFOPNSF) API with your file system, you must:

1. Write an exit program that performs the open stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Open Stream File (QHFOPNSF) API" on page 14-13.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFOPNSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

Operation (OPNSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (OPNSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Open file handle
OUTPUT; CHAR(16)

Path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Path name length
INPUT; BINARY(4)

Open information
INPUT; CHAR(10)
The exit program can ignore character 7, which describes the type of open operation to perform. This field is used by OS/400 HFS support during job cleanup if the job ends before the file is closed.

Attribute information table
INPUT; CHAR(*)

Length of the attribute information table
INPUT; BINARY(4)

Action taken
OUTPUT; CHAR(1)

API Functions: The QHFOPNSF API performs the standard functions described on page 15-2 and these additional functions:

- Verifies that the length of the attribute information table is not negative.
- Ensures that the file is closed during normal job cleanup, in case the user forgets to close it before ending the job.
- Handles the type-of-open value represented by character 7 of the open information parameter. The file system does not need to take any action on the basis of this value.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Checks for previous open operations that have lock modes conflicting with the requested access mode for this file.
- Attempts to take the action designated by the open information. The action can be opening an existing file, opening and replacing an existing file, or creating and opening a new file.
- If the action is successful, assigns a file handle, locks the file and its attributes, and returns the handle and action taken to the API. The API does not return this handle to the application. The API creates a handle of its own to return to the application. This procedure improves API performance and ensures that handles are unique across file systems.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F01 E Directory name not valid.
CPF1F02 E Directory not found.
CPF1F06 E Directory in use.
CPF1F07 E Authority not sufficient to access directory.
CPF1F08 E Damaged directory.
CPF1F2A E Number of open files exceeds limit.
CPF1F21 E File name not valid.
CPF1F22 E File not found.
CPF1F24 E File name already exists.
CPF1F26 E File in use.
CPF1F27 E Authority not sufficient to access file.
CPF1F28 E Damaged file.
CPF1F29 E Use of reserved file name not allowed.
CPF1F37 E File is a read-only file.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F42 E Attribute information table not valid.
CPF1F43 E Attribute name not valid.
CPF1F44 E Attribute value is not valid.
CPF1F46 E Use of reserved attribute name not allowed.
CPF1F48 E Path name not valid.
CPF1F49 E Open information value not valid.
CPF1F61 E No free space available on media.
CPF1F62 E Requested function failed.
CPF1F63 E Media is write protected.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F75 E Error occurred during start-job-session function.
CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Read Directory Entries (QHFRDDR) API

Parameters

Required Parameter Group:

1	Operation (RDDR)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Open directory handle	Input	Char(16)
4	Data buffer	Output	Char(*)
5	Data buffer length	Input	Binary(4)
6	Number of directory entries to read	Input	Binary(4)
7	Number of directory entries read	Output	Binary(4)
8	Length of data returned	Output	Binary(4)

Before applications can use the Read Directory Entries (QHFRDDR) API with your file system, you must:

1. Write an exit program that performs the read directory entries operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Read Directory Entries (QHFRDDR) API" on page 14-16.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRDDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs

the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameter that the exit program must pass back to the API:

Operation (RDDR)
INPUT; CHAR(5)
The abbreviation for the operation being performed (RDDR).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

Open directory handle
INPUT; CHAR(16)

Data buffer
OUTPUT; CHAR(*)

Data buffer length
INPUT; BINARY(4)

Number of directory entries to read
INPUT; BINARY(4)

Number of directory entries read
OUTPUT; BINARY(4)

Length of data returned
OUTPUT; BINARY(4)

API Functions: The QHFRDDR API performs the standard functions described on page 15-2 and one additional function. The API validates the open directory handle to ensure that the directory is open and the current user profile is the user that opened it.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Retrieves the directory entry information. The file system should return only attributes selected with the attribute selection table when the directory was opened. The file system must build the table and set the number of directory entries actually read and the length of data returned.
- If a requested attribute is not associated with a directory entry, returns the attribute name and the length of the attribute value, which is zero.
- Increases the directory pointer value to reflect its new position after directory entries are read.

In addition, your file system's documentation should describe the order in which directory entries are returned (for example, alphabetic or last-used date).

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F08 E Damaged directory.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F47 E Buffer overflow occurred.
CPF1F53 E Value for length of data buffer not valid.
CPF1F62 E Requested function failed.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Read from Stream File (QHFRDSF) API

Parameters

Required Parameter Group:

1	Operation (RDSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Open file handle	Input	Char(16)
4	Data buffer	Output	Char(*)
5	Bytes to read	Input	Binary(4)
6	Bytes actually read	Output	Binary(4)

Before applications can use the Read from Stream File (QHFRDSF) API with your file system, you must:

1. Write an exit program that performs the read operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Read from Stream File (QHFRDSF) API" on page 14-18.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRDSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

Operation (RDSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (RDSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

Open file handle
INPUT; CHAR(16)

Data buffer
OUTPUT; CHAR(*)

Bytes to read
INPUT; BINARY(4)

Bytes actually read
OUTPUT; BINARY(4)

API Functions: The QHFRDSF API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Verifies that the file was previously opened with an access mode of read only or read/write.
- Checks the range of bytes being read to make sure no part of the range is locked in deny read/write mode, which would preclude this operation.
- Reads the number of bytes specified from the file, starting at the current file pointer position, and places the data read in the data buffer.
- Records the number of bytes actually read. If the end of the file is reached during the read operation, this number is less than the number specified in the bytes-to-read parameter.
- Increases the value of the file pointer by the number of bytes read.
- If the read operation is not successful, sets the bytes actually read to zero and returns an exception describing the error to the API.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F2C E	Read operation not allowed to file opened for write only.
CPF1F2E E	Range of bytes in file in use.
CPF1F28 E	Damaged file.
CPF1F35 E	Read file operation failed.
CPF1F4B E	Value for number of bytes not valid.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F62 E	Requested function failed.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program for Rename Directory (QHFRNMDR) API

Parameters

Required Parameter Group:

1	Operation (RNMDR)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Path name	Input	Char(*)
5	Path name length	Input	Binary(4)
6	New directory name	Input	Char(*)
7	New directory name length	Input	Binary(4)

Before applications can use the Rename Directory (QHFRNMDR) API with your file system, you must:

1. Write an exit program that performs the rename directory operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Rename Directory (QHFRNMDR) API" on page 14-19.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRNMDR API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (RNMDR)
INPUT; CHAR(5)
The abbreviation for the operation being performed (RNMDR).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Path name length
INPUT; BINARY(4)

New directory name
INPUT; CHAR(*)

New directory name length
INPUT; BINARY(4)

API Functions: The QHFRNMDR API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Verifies that the new directory does not already exist and that it has a different name from the old directory.
- Verifies that the directory is not in use before renaming it.
- Renames the directory.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F03 E	New directory name same as old directory name.
CPF1F04 E	Directory name already exists.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.
CPF1F08 E	Damaged directory.
CPF1F09 E	Use of reserved directory name not allowed.
CPF1F41 E	Severe error occurred while addressing parameter list.
CPF1F48 E	Path name not valid.
CPF1F61 E	No free space available on media.
CPF1F62 E	Requested function failed.
CPF1F63 E	Media is write protected.
CPF1F66 E	Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E	Exception specific to file system occurred.
CPF1F73 E	Not authorized to use command.
CPF1F74 E	Not authorized to object.
CPF1F75 E	Error occurred during start-job-session function.
CPF1F77 E	Severe parameter error occurred on call to file system.

Exit Program for Rename Stream File (QHFRNMSF) API

Parameters

Required Parameter Group:

1	Operation (RNMSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Path name	Input	Char(*)
5	Path name length	Input	Binary(4)
6	New file name	Input	Char(*)
7	New file name length	Input	Binary(4)

Before applications can use the Rename Stream File (QHFRNMSF) API with your file system, you must:

1. Write an exit program that performs the rename stream file operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Rename Stream File (QHFRNMSF) API" on page 14-19.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRNMSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (RNMSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (RNMSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Path name length
INPUT; BINARY(4)

New file name
INPUT; CHAR(*)

New file name length
INPUT; BINARY(4)

API Functions: The QHFRNMSF API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Verifies that the new file name does not exist and is different from the current file name.
- Ensures that the file being renamed is not open or in use.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F01 E	Directory name not valid.
CPF1F02 E	Directory not found.
CPF1F06 E	Directory in use.
CPF1F07 E	Authority not sufficient to access directory.

| CPF1F08 E Damaged directory.
| CPF1F21 E File name not valid.
| CPF1F22 E File not found.
| CPF1F23 E New file name same as old file name.
| CPF1F24 E File name already exists.
| CPF1F26 E File in use.
| CPF1F27 E Authority not sufficient to access file.
| CPF1F28 E Damaged file.
| CPF1F29 E Use of reserved file name not allowed.
| CPF1F41 E Severe error occurred while addressing parameter list.
| CPF1F48 E Path name not valid.
| CPF1F61 E No free space available on media.
| CPF1F62 E Requested function failed.
| CPF1F63 E Media is write protected.
| CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
| CPF1F71 E Exception specific to file system occurred.
| CPF1F73 E Not authorized to use command.
| CPF1F74 E Not authorized to object.
| CPF1F75 E Error occurred during start-job-session function.
| CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Retrieve Directory Entry Attributes (QHFRTVAT) API

Parameters

Required Parameter Group:

1	Operation (RTVAT)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Reserved	Input	Char(20)
4	Path name	Input	Char(*)
5	Path name length	Input	Binary(4)
6	Attribute selection table	Input	Char(*)
7	Length of attribute selection table	Input	Binary(4)
8	Attribute information table	Output	Char(*)
9	Length of attribute information table	Input	Binary(4)
10	Length of data returned	Output	Binary(4)

Before applications can use the Retrieve Directory Entry Attributes (QHFRTVAT) API with your file system, you must:

1. Write an exit program that performs the retrieve attributes operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Retrieve Directory Entry Attributes (QHFRTVAT) API" on page 14-20.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFRTVAT API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

Operation (RTVAT)
INPUT; CHAR(5)
The abbreviation for the operation being performed (RTVAT).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

Reserved
INPUT; CHAR(20)
Reserved for future use. This parameter is set to blanks.

Except as noted, the following parameters are the same as the parameters for the API.

Path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

Path name length
INPUT; BINARY(4)

Attribute selection table
INPUT; CHAR(*)

Length of the attribute selection table
INPUT; BINARY(4)

Attribute information table
OUTPUT; CHAR(*)

Length of the attribute information table
INPUT; BINARY(4)

Length of data returned
OUTPUT; BINARY(4)

API Functions: The QHFRTVAT API performs the standard functions described on page 15-2. The API does not validate the attribute selection table, the attribute information table, or the length of the attribute information table.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Validates the attribute selection table.
- Builds the attribute information table to return the requested attributes to the application.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

| CPF1F01 E Directory name not valid.
| CPF1F02 E Directory not found.
| CPF1F06 E Directory in use.

| CPF1F07 E Authority not sufficient to access directory.
| CPF1F08 E Damaged directory.
| CPF1F21 E File name not valid.
| CPF1F22 E File not found.
| CPF1F26 E File in use.
| CPF1F27 E Authority not sufficient to access file.
| CPF1F28 E Damaged file.
| CPF1F41 E Severe error occurred while addressing parameter list.
| CPF1F42 E Attribute information table not valid.
| CPF1F43 E Attribute name not valid.
| CPF1F45 E Attribute selection table not valid.
| CPF1F47 E Buffer overflow occurred.
| CPF1F48 E Path name not valid.
| CPF1F62 E Requested function failed.
| CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
| CPF1F71 E Exception specific to file system occurred.
| CPF1F73 E Not authorized to use command.
| CPF1F74 E Not authorized to object.
| CPF1F75 E Error occurred during start-job-session function.
| CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Set Stream File Size (QHFSETSZ) API

Parameters

Required Parameter Group:

1	Operation (SETSZ)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Open file handle	Input	Char(16)
4	File size	Input	Binary(4) Unsigned

Before applications can use the Set Stream File Size (QHFSETSZ) API with your file system, you must:

1. Write an exit program that performs the set size operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Set Stream File Size (QHFSETSZ) API" on page 14-21.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFSETSZ API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The API passes this information to your exit program:

Operation (SETSZ)
INPUT; CHAR(5)
The abbreviation for the operation being performed (SETSZ).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

Open file handle
INPUT; CHAR(16)

File size
INPUT; BINARY(4) UNSIGNED

API Functions: The QHFSETSZ API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Checks for byte locks that conflict with changing the size of the file, and returns an exception if any are found. The application cannot set the file size into or beyond a locked range.
- Verifies that the file size parameter value is valid for that file system.
- Increases or decreases the file size.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

| CPF1F2B E Write operation not allowed to file opened for read only.
| CPF1F2E E Range of bytes in file in use.
| CPF1F28 E Damaged file.
| CPF1F4B E Value for number of bytes not valid.
| CPF1F41 E Severe error occurred while addressing parameter list.
| CPF1F61 E No free space available on media.
| CPF1F62 E Requested function failed.
| CPF1F63 E Media is write protected.
| CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
| CPF1F71 E Exception specific to file system occurred.
| CPF1F73 E Not authorized to use command.
| CPF1F74 E Not authorized to object.
| CPF1F77 E Severe parameter error occurred on call to file system.

Exit Program for Write to Stream File (QHFWRTSF) API

Parameters

Required Parameter Group:

1	Operation (WRTSF)	Input	Char(5)
2	File system job handle	Input	Char(16)
3	Open file handle	Input	Char(16)
4	Data buffer	Input	Char(*)
5	Bytes to write	Input	Binary(4)
6	Bytes actually written	Output	Binary(4)

Before applications can use the Write to Stream File (QHFWRTSF) API with your file system, you must:

1. Write an exit program that performs the write operation on behalf of the API. For a detailed description of the API and its calling parameters, see "Write to Stream File (QHFWRTSF) API" on page 14-22.
2. Give the exit program's name when you register the file system with the Register File System (QHFRGFS) API.

After that, when an application calls the QHFWRTSF API, the API calls your exit program and passes it the parameters specified by the application. Your exit program performs the work and returns any data to the API. The API passes the data back to the calling application.

Required Parameter Group: The following shows the input parameters that the API passes to your exit program and the output parameters that the exit program must pass back to the API:

Operation (WRTSF)
INPUT; CHAR(5)
The abbreviation for the operation being performed (WRTSF).

File system job handle
INPUT; CHAR(16)
The work area or job identifier for use by the file system.

The following parameters are the same as the parameters for the API.

Open file handle
INPUT; CHAR(16)

Data buffer
INPUT; CHAR(*)

Bytes to write
INPUT; BINARY(4)

Bytes actually written
OUTPUT; BINARY(4)

API Functions: The QHFWRTSF API performs the standard functions described on page 15-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 15-3 and these additional functions:

- Verifies that the file was previously opened with an access mode of write or read/write.
- Makes sure that no part of the range of bytes being written is locked in a way that denies access to this operation.
- Writes the number of bytes specified in the data buffer to the file, starting at the current file pointer position.
- Records the number of bytes actually written. Unless an error occurs, this number must be the same as the number specified in the bytes-to-write parameter.
- Increases the value of the file pointer by the number of bytes written.
- If the write operation is not successful, sets the bytes actually written to zero and signals an exception describing the error to the API.

Error Messages for Exit Program Use: This section lists the messages that the exit program can return to the API.

CPF1F2B E Write operation not allowed to file opened for read only.
CPF1F2E E Range of bytes in file in use.
CPF1F28 E Damaged file.
CPF1F34 E Attempted write operation beyond file size limit.
CPF1F36 E Write file operation failed.
CPF1F4B E Value for number of bytes not valid.
CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F61 E No free space available on media.
CPF1F62 E Requested function failed.
CPF1F63 E Media is write protected.
CPF1F66 E Storage needed exceeds maximum limit for user profile &1.
CPF1F71 E Exception specific to file system occurred.
CPF1F73 E Not authorized to use command.
CPF1F74 E Not authorized to object.
CPF1F77 E Severe parameter error occurred on call to file system.

Part 6. High-Level Language APIs

Chapter 16. Application Development Manager/400 APIs

The SAA* AD/Cycle* Application Development Manager/400 APIs allow a control language (CL) command such as the Build Part, BLDPART, command to determine, for example, the includes and external references that were used by certain processors when processing a source member. The term **processor** is used in these APIs to mean compiler or preprocessor.

In Application Development Manager/400 terms, a part can be either a source member or an object, such as a file. Refer to the appropriate Application Development Manager/400 publication, as listed in the bibliography, for more information.

If you have an application that can use the information provided by the APIs, you can call these APIs from any high-level programming language. The Application Development Manager/400 product does not need to be installed on your system for you to use these APIs.

The APIs are presented in alphabetical order and include:

- Get Space Status (QLYGETS)
- Read Build Information (QLYRDBI)
- Set Space Status (QLYSETS)
- Write Build Information (QLYWRTBI)

The Get and Set Status APIs are used to query and initialize the build information space that is to contain the Application Development Manager/400 information. The Write and Read Build Information APIs are used to write or read records of build information to and from the space.

There are several different types of records that can be read or written using the Application Development Manager/400 APIs. These record types are explained in "Record Types" on page 16-4.

The following compilers and preprocessors use these APIs.

Figure 16-1. Compilers and Preprocessors That Interface with the Application Development Manager/400 Product

Compiler/ Pre-processor	Compiler/Preprocessor OS/400 Command	N-1 Support
RPG/400	CRTRPGPGM	Yes
COBOL/400	CRTCBLPGM	Yes
C/400	CRTCPGM	No
DDS	CRTPF, CRTLF, CRTDSPF, CRTPRTF, CRTICFF	N/A
CL	CRTCLPGM	Yes
CLD	CRTCLD	Yes
CMD	CRTCMD	N/A
SQL/400	CRTSQLRPG, CRTSQLCBL, CRTSQLC	Yes

The following diagram shows the proper usage and order in which the APIs should be called.

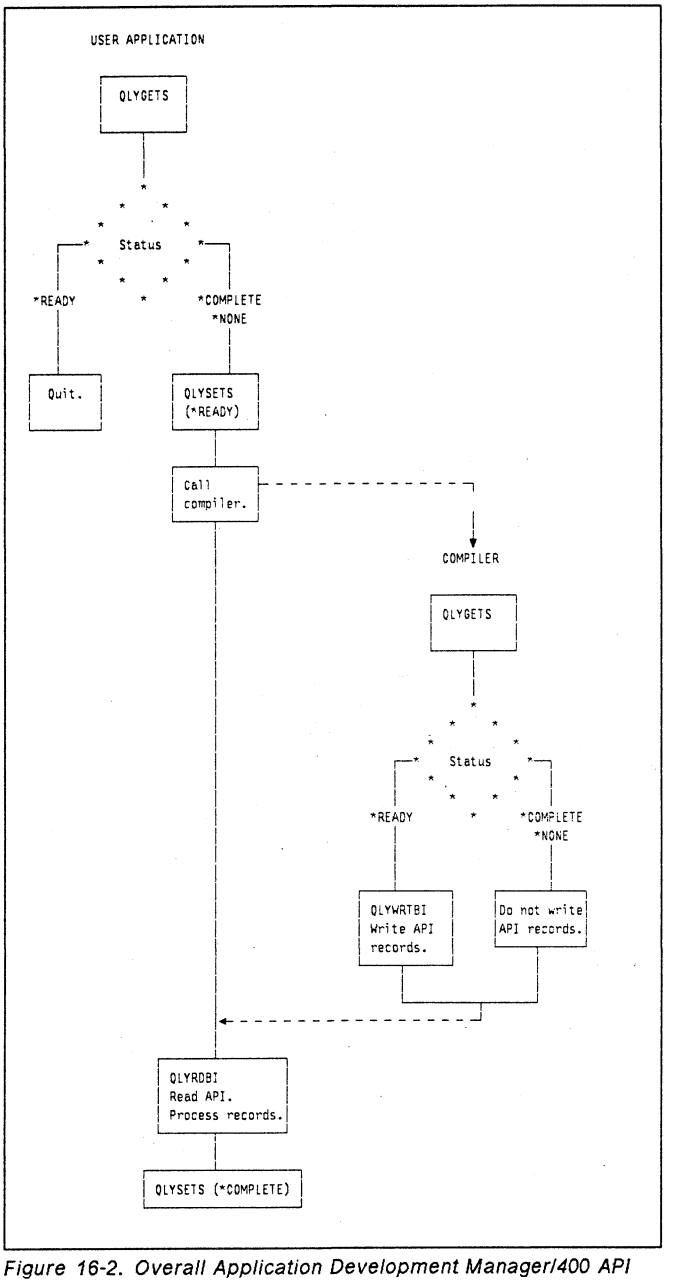


Figure 16-2. Overall Application Development Manager/400 API Usage

QLYGETS should be called by the application or compiler prior to calling the other three APIs, QLYSETS, QLYWRTBI, and QLYRDBI, to verify that the space is available for use.

The following table describes the API space status values that can be received by calling the QLYGETS API, and the action that should be taken by the application or compiler that is calling the API.

Figure 16-3. API Space Status

Status	Application	Compiler
*COM-PLETE	The space is available for use. Call QLYSETS to set to *READY.	Do not write API records.
*NONE	The space does not exist. The application calls QLYSETS to create and set the space to *READY.	Do not write API records.
*READY	The space is in use by a compiler. The other APIs should not be called.	The space is available for writing.

Compilers use the APIs to write to the space. Applications use the APIs to read from the space.

Note: Unpredictable results can occur when the APIs are not properly used or are used in the incorrect order. Calling multiple API-supporting compilers simultaneously in a single interactive session (one possible way of doing this is by pressing the Attention key and then command key F9 to get to the command line) may cause unpredictable results. The compiler can fail, for example, or incorrect or incomplete information can be put in the work space.

Get Space Status (QLYGETS) API

Parameters

Required Parameter Group:

1	Status	Output	Char(10)
2	Error code	I/O	Char(*)

The Get Space Status (QLYGETS) API obtains the status of the space.

Required Parameter Group

Status
OUTPUT; CHAR(10)

***READY**	Information in the space is ready to be processed.
***COMPLETE**	Information in the space has been processed.
***NONE**	The space does not exist. Use QLYSETS to create the space.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF3CF1 Error code parameter not valid.

Read Build Information (QLYRDBI) API

Parameters

Required Parameter Group:

1	Buffer	Output	Char(*)
2	Maximum size	Input	Binary(4)
3	Read mode	Input	Char(10)
4	Buffer length	Output	Binary(4)
5	Number of records	Output	Binary(4)
6	Error code	I/O	Char(*)

The Read Build Information (QLYRDBI) API reads one or more records from the space.

QLYRDBI reads the space starting at the first location after the last record was read. If this is the first time QLYRDBI is called, the first record following the header record is read.

After QLYRDBI has read the Normal processor end record or the Abnormal processor end record, the next call to QLYRDBI starts reading the space from the beginning again.

QLYRDBI reads one or more records depending on the value specified on the Read-mode parameter. QLYRDBI does not read more records than can fit in the buffer. The buffer is determined by the Maximum-size parameter.

Required Parameter Group

Buffer
OUTPUT; CHAR(*)
A character string to contain one or more records of build information.

Maximum size
INPUT; BINARY(4)
The maximum size of the data that is expected to be returned to this call. Maximum size should be large enough to fit at least one record. If it is too small for one record, an error occurs.

Read mode
INPUT; CHAR(10)
The mode of reading. The possible Read-mode values are:

***SINGLE**	Read only one record.
***MULTIPLE**	Read more than one record. The maximum number of records that are read is determined by the size of Maximum size.

Buffer length
OUTPUT; BINARY(4)
The length of the data returned. If records are not read, 0 is returned.

Number of records
OUTPUT; BINARY(4)
The number of records read. Number of records is 0 if no records were read, 1 if one record was read or

greater than 1 if *MULTIPLE was specified on Read mode and more than one record could fit in the buffer.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

LIB9005	Value specified for the maximum-size parameter is not valid.[1]
LIB9006	Value specified for the read-mode parameter is not valid.
LIB9007	The value specified for the maximum-size parameter is too small.
LIB9009	Space does not exist, or it is damaged or deleted.
LIB9010	Build information missing or no more build information.
LIB9011	Build information in the space is not complete.
CPF3CF1	Error code parameter not valid.

Set Space Status (QLYSETS) API

Parameters

Required Parameter Group:

1	Status	Input	Char(10)
2	Error code	I/O	Char(*)

The Set Space Status (QLYSETS) API sets the status of the space.

When QLYSETS is first called to create the space (if the space does not exist already) or to initialize the space so the information can be written to it by compilers or preprocessors, the Status parameter should be set to *READY. Then QLYSETS writes a special record (called the HEADER record) at the beginning of the space and initializes a status flag in that record to *READY. Now the space is ready to accept records containing build information. Compilers write to the space using the QLYWRTBI API. QLYWRTBI writes records to the space concatenated to each other. QLYRDBI later reads them sequentially in the order in which they are written.

Use the QLYSETS API to set the status flag in the space to *COMPLETE after the information in the space is processed using the QLYRDBI API. This indicates that the information in the space has been processed and the space can be reused.

Required Parameter Group

Status
INPUT; CHAR(10)
The status for the space. The possible status values are:

***READY**	Initialize the space. If the space does not exist, it is created.
***COMPLETE**	Information in the space has been processed. The space can now be used by setting it to *READY with another call to QLYSETS.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

LIB9001	Value specified on the status parameter is not valid.
CPF3CF1	Error code parameter not valid.

Write Build Information (QLYWRTBI) API

Parameters

Required Parameter Group:

1	Buffer	Input	Char(*)
2	Buffer length	Input	Binary(4)
3	Error code	I/O	Char(*)

The Write Build Information (QLYWRTBI) API writes one or more records to the space.

QLYWRTBI writes records to the space concatenated to each other. QLYRDBI later reads them sequentially in the order in which they are written.

After QLYWRTBI writes the Normal processor end record or the Abnormal processor end record, the next call to QLYWRTBI starts writing records from the beginning of the space again. See "Record Types" on page 16-4 for the records that can be written.

Required Parameter Group

Buffer
INPUT; CHAR(*)
A character string containing one or more records of build information.

Buffer length
INPUT; BINARY(4)
The length of the buffer in bytes. The buffer length must be equal to the sum of the lengths of all the concatenated records being passed, otherwise an error occurs.

[1] The LIBxxxx error messages are located in the message file QLIBMSG in the QSYS library.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

The first field in each record indicates the record length. This allows all the records to be read sequentially using the QLYRDBI API.

Error Messages

LIB9002	Value specified for the buffer length parameter is not valid.
LIB9003	The value specified for the buffer length parameter is too small.
LIB9004	Record not in correct sequence.
LIB9008	Record has a record type that is not valid.
LIB9009	Space does not exist, or it is damaged or deleted.
CPF3CF1	Error code parameter not valid.

Record Types

This section describes the information contained in all the different record types. Typically a compiler writes records and an application reads them.

Names, field types and other information passed through the different record types are *not* validated and no authority is checked by QLYWRTBI. The QLYWRTBI API assumes that all that validation and checking has been done.

There are the following record types:

- Processor start record
- Normal processor end record
- Normal processor end call next record
- Abnormal processor end record
- Include record
- File reference record
- Record format reference record
- Field reference record
- Message reference record
- External reference error record
- Object already exists error record
- Start of new program record.

The following table shows the records that can be written by each compiler.

The following is true for the *Library specified* fields for all records and compilers:

- When *CURLIB is specified for the *Library specified* fields, *CURLIB is passed.
- When *LIBL is specified for the *Library specified* fields, or implied by not being specified, *LIBL is passed.

Notes and restrictions are explained in the footnotes following the table.

Figure 16-4. Record Types and Processors

Record Type	RPG/400	COBOL/400	C/400	CLD	DDS	CL(5)	CMD	SQL/400
Processor start	X(1 3)	X	X(3)	X(1 3)	X	X(1 3)	X(1)	X(1)
Normal processor end	X	X	X	X	X	X	X	X
Normal processor end call next								X
Abnormal processor end	X	X	X	X	X	X	X	X
Include	X(11)	X	X(8 12)					X(1 7)
File reference	X	X	X		X	X(1)		X(1)
Record format reference	X	X	X		X	X(1)		X(1)
Field reference					X(2)			
Message reference					X(2 9)		X (1 2 6 9)	
External reference error	X	X	X(10 13)		X(10)	X(1 4)		X(1)
Object already exists error					X			
Start of new program		X						

Notes and Restrictions for Table Above:

1. If *CURLIB is specified for the *Library specified* fields (this includes the *Source library specified* field on the Processor start record), the resolved library name is passed instead of *CURLIB.
2. If *LIBL is specified for the *Library specified* fields, or implied by not being specified, the resolved library name is passed instead of *LIBL.
3. If *CURLIB is specified for the *Target library* field, the resolved library name is passed instead of *CURLIB.
4. For most *Used* fields, when a file being referred to on the DCLF command cannot be found, CL puts blanks in this field. There is no actual file or library name when the file is not found.
5. For all fields marked *Reserved*, CL initializes them to hex zeros. However, fields that are not reserved are set to blanks when they do not apply and are defined as character. For example, *Target member* on the Processor Start record does not have meaning for the CL compiler and is initialized to blanks.
6. Message reference records are written only for messages specified on the PROMPT parameter of the PARM, ELEM, or QUAL command definition statement.
7. The SQL/400 compiler does not write include records for the following statements:

   EXEC SQL INCLUDE SQLCA
   EXEC SQL INCLUDE SQLDA

   These statements are not true includes in the sense that the SQL/400 compiler does not read source from another member or source file.
8. The C/400 compiler does not write API Include records for system include files. File names enclosed in angle brackets, (< ... >), designate system include files. File names enclosed in double quotation marks, (" ... "), designate user include files.
9. The *Message file used* and *Library used* fields are always blank.
10. If *LIBL is specified in the source, or implied by not being specified (*Library specified* is *LIBL), the *Library used* field will be set to *LIBL because no specific library can be determined if the file is not found in the library list.
11. RPG puts *LIBL in the *Library specified* field if it is not specified, and QRPGSRC in the *File specified* field if it is not specified.
12. The *Library specified* field is the resolved library name if the library name is not specified. The *Include file specified* field will contain the resolved file name if the file name is not specified.
13. If *CURLIB is specified in the source (*Library specified* is *CURLIB), the *Library used* field will be set to *CURLIB because no specific library can be determined if the file is not found in the library list.

Processor Start Record

This must be the first record that is passed by the compiler or preprocessor on its first call to the QLYWRTBI API.

The Processor start record has the following format:

Figure 16-5. Processor Start Record

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(10)	Processor command
18	12	CHAR(10)	Source object name specified
28	1C	CHAR(10)	Source library name specified
38	26	CHAR(7)	Source object type
45	2D	CHAR(10)	Source member name specified
55	37	CHAR(10)	Source object name used
65	41	CHAR(10)	Source library name used
75	4B	CHAR(10)	Source member name used
85	55	CHAR(10)	Target object name specified
95	5F	CHAR(10)	Target library name specified
105	69	CHAR(7)	Target object type
112	70	CHAR(10)	Target member name specified
122	7A	CHAR(2)	Reserved

Processors for which this record type is applicable: All compilers and preprocessors listed in Figure 16-1 on page 16-1.

Field Descriptions

Processor command. The compiler or preprocessor that wrote this record (for example, CRTRPGPGM).

Record length. The length of this record is 124.

Record type. The type of this record is '01'.

Reserved. An ignored field.

Source library name used. The actual name of the library that was used. The library name could be different from the specified library name because *LIBL or *CURLIB was specified, or an override was used. This field contains the name the library resolves to.

Source library name specified. The library name of the source file specified on the compiler or preprocessor command.

Source member name used. The actual name of the source member that was used. This field is required, even if the two member names are the same.

Source member name specified. The source member name specified on the compiler or preprocessor command.

Source object name used. The actual name of the object that was used. The object name could be different from the specified object name if an override was used.

Source object name specified. The object name specified on the compiler or preprocessor command.

Source object type. The AS/400 type of the source object (for example, *FILE.)

Target library name specified. The library of the target object specified on the compiler or preprocessor command.

Target member name specified. The name of the member to be created, if applicable, specified on the compiler or preprocessor command.

Target object name specified. The name of the object to be created, called the target object, specified on the compiler or preprocessor command. The actual name of the object that was created is passed through the Normal processor end record. (See "Normal Processor End Record.")

Target object type. The AS/400 type of the object to be created (for example, *FILE.)

Normal Processor End Record

This is the last record passed by the compiler or preprocessor to indicate that processing ended successfully.

The Normal processor end record has the following format:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(10)	Object name created
18	12	CHAR(10)	Library name
28	1C	CHAR(7)	Object type
35	23	CHAR(10)	Member name
45	2D	CHAR(7)	Message identifier

Processors for which this record type is applicable: All compilers and preprocessors listed in Figure 16-1 on page 16-1.

Field Descriptions

Library name. The library where the object was created.

Member name. The name of the member created, if applicable.

Message identifier. The message identification of the completion message.

Object name created. The object created by the compiler or preprocessor. If an object is not created, this field stores the value of '*NONE'.

Object type. The type of object created.

Record length. The length of this record is 52.

Record type. The type of this record is '20'.

Reserved. An ignored field.

Normal Processor End Call Next Record

When a preprocessor successfully creates an object or member and needs to call another compiler or preprocessor, it should pass this record instead of passing the Normal processor end record as the final record. For example, if the CRTSQLRPG command is entered with OPTION(*GEN), and the member is created successfully, the last record written by CRTSQLRPG is the Normal processor end call next record. The preprocessor then calls CRTRPGPGM that eventually writes the normal or abnormal record.

The Normal processor end call next record has the following format:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(10)	Object name
18	12	CHAR(10)	Library name
28	1C	CHAR(7)	Object type
35	23	CHAR(10)	Member name
45	2D	CHAR(7)	Message identifier

Processors for which this record type is applicable: The SQL/400 preprocessor if OPTION(*GEN) is specified on the command.

Field Descriptions

Library name. The library where the object was created.

Member name. The name of the member created, if applicable.

Message identifier. The message identification of the completion message.

Object name. The name of the object created.

Object type. The type of object created.

Record length. The length of this record is 52.

Record type. The type of this record is '21'.

Reserved. An ignored field.

Abnormal Processor End Record

This is the last record passed if the compiler or preprocessor fails due to an error. For example, an object or member was not created due to compile errors, or REPLACE(*NO) was specified on the command and the object existed.

If the command failed because an external reference to a file or message file could not be found, the command passes the External reference error record before passing this one. See "External Reference Error Record" on page 16-10 for more information on this record.

The Abnormal processor end record has the following format:

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(7)	Message identifier
15	F	CHAR(1)	Reserved

Processors for which this record type is applicable: All compilers and preprocessors listed in Figure 16-1 on page 16-1.

Field Descriptions

Message identifier. The message identification of the completion message.

Record length. The length of this record is 16.

Record type. The type of this record is '30'.

Reserved. An ignored field.

Include Record

This record is passed when the compiler or preprocessor processes an include. An **include** statement is a statement that causes the compiler to replace the include statement with the contents of the specified header or file. If the include is not found, the compiler or preprocessor passes the Abnormal processor end record.

The Include record has the following format:

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	BINARY(4)	Nesting level
12	C	CHAR(10)	Include file name specified
22	16	CHAR(10)	Include file library name specified
32	20	CHAR(10)	Include file member name specified
42	2A	CHAR(7)	Object type
49	31	CHAR(10)	Include file name used
59	3B	CHAR(10)	Include file library name used
69	45	CHAR(10)	Include file member name used
79	4F	CHAR(1)	Reserved

Processors for which this record type is applicable:
SQL/400, RPG/400, COBOL/400, C/400 processors.

Field Descriptions

Include file used. The actual name of the include file that was used. For example, the default include file used by the compiler and implied in the source, or the file different than the one specified in the source as a result of an override. This name must always be filled in.

Include file specified. The name of the file that contains the include. This is the name specified in the source (if the include was file qualified), otherwise it is blank.

Include file library used. The name of the actual library that contains the include file that was used (for example, a specific library name instead of *CURLIB or *LIBL, as specified in the source, or a library different than the one specified in the source, as a result of an override).

Include file library specified. The name of the library where the include file resides, as specified in the source (if the include was library qualified), otherwise it is blank.

Include file member used. The actual name of the source member containing the include that was used. This name must always be filled in.

Include file member specified. The name of the source member containing the include, as specified in the source.

Nesting level. The level of nesting of the include. Includes found in the root source have a nesting level of 1, includes found in level 1 have a nesting level of 2 and so on.

Object type. The object type of the object containing the include, for example *FILE.

Record length. The length of this record is 80.

Record type. The type of this record is '02'.

Reserved. An ignored field.

The nesting level should be indicated even by those compilers that do not allow include nesting. In that case, the nesting level passed should be equal to 1.

File Reference Record

This record is passed when the compiler or preprocessor encounters a reference to an externally described file but not its record format or field.

For example, a reference is made in DDS source via the PFILE or JFILE keywords. Another example is when a compiler or preprocessor copies *all* the record format declares from a file. This is not considered to be a dependency on any specific record format and is treated as a dependency on the file, so this record must be passed, not the Record format reference records for all the individual record formats.

The File reference record has the following format:

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(10)	File name specified
18	12	CHAR(10)	File library name specified
28	1C	CHAR(1)	Based on indicator
29	1D	CHAR(10)	File name used
39	27	CHAR(10)	File library name used
49	31	CHAR(3)	Reserved
52	34	BINARY(4)	Nesting level

Processors for which this record type is applicable:
SQL/400, DDS, CL, RPG/400, COBOL/400, C/400 processors.

Field Descriptions

Based on indicator. Indicates whether the referenced file is used to base another file on. Possible values are "N" (no) and "Y" (yes).

File name used. The name of the actual file that was referenced. This name must always be filled in.

File name specified. The name of the file referenced, as specified in the source.

File library name used. The name of the actual library that contains the file that was referenced. The library name could be different from the specified library name because *LIBL or *CURLIB was specified, or an override was used.

File library name specified. The name of the library of the file referenced, as specified in the source.

Nesting level. If this file reference is made within an include, this field has value of N + 1, where N is the nesting level of the include. Otherwise, the value of this field is 1.

Record length. The length of this record is 56.

Record type. The type of this record is '03'.

Reserved. An ignored field.

Record Format Reference Record

This record is passed when the compiler or preprocessor encounters a reference to a record format of an externally described file (but not to any single field). For example, a reference is made in DDS source via the FORMAT keyword or in the RPG/400, COBOL/400, C/400, CL or SQL/400 processors whenever a declaration of a record format structure from a DDS-described file is generated by the compiler or preprocessor.

The Record format reference record has the following format:

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(10)	File name specified
18	12	CHAR(10)	File library name specified
28	1C	CHAR(10)	Record format name
38	26	CHAR(13)	Record format level ID
51	33	CHAR(10)	File name used
61	3D	CHAR(10)	File library name used
71	47	CHAR(1)	Reserved
72	48	BINARY(4)	Nesting level

Processors for which this record type is applicable:
SQL/400, DDS, CL, RPG/400, COBOL/400, C/400 processors.

Field Descriptions

File name used. The name of the actual file that was referenced. This name must always be filled in.

File name specified. The name of the file being referenced, as specified in the source.

File library name used. The name of the actual library that contains the file that was referenced. The library name could be different from the specified library name because *LIBL or *CURLIB was specified, or an override was used. This field contains the name the library resolves to.

File library name specified. The name of the library of the file being referenced, as specified in the source.

Nesting level. If this record format reference is made within an include, this field has value of N + 1, where N is the nesting level of the include. Otherwise, the value of this field is 1.

Record format level ID. The level ID of the record format referenced.

Record format name. The name of the record format referenced.

Record length. The length of this record is 76

Record type. The type of this record is '04'.

Reserved. An ignored field.

Field Reference Record

This record is passed when the compiler or preprocessor encounters a reference to a field in an externally described file. For example, a reference is made in DDS source via the REF and REFFLD keywords.

The Field reference record has the following format:

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(10)	File name specified
18	12	CHAR(10)	File library name specified
28	1C	CHAR(10)	Record format name
38	26	CHAR(13)	Record format level ID
51	33	CHAR(10)	Field
61	3D	CHAR(3)	Reserved
64	40	BINARY(4)	Field length
68	44	BINARY(4)	Decimal positions
72	48	CHAR(1)	Data type
73	49	CHAR(1)	Fixed/variable length indicator
74	4A	CHAR(10)	File name used
84	54	CHAR(10)	File library name used
94	5E	CHAR(2)	Reserved

Processors for which this record type is applicable: DDS.

Field Descriptions

Data type. The field data type in DDS. For example, P, S, B, F, A, or H.

Decimal positions. The number of decimal positions if the field is numeric, otherwise 0.

Field. The name of the referenced field.

Field length. The length of the field in bytes. If the field is a variable-length field, the maximum length should be passed.

File name used. The name of the actual file that was referenced. This name must always be filled in.

File name specified. The name of the file being referenced, as specified in the source.

Fixed/variable length indicator. Contains F if the field is of fixed length, V if variable length.

File library name used. The name of the actual library that contains the file that was referenced.

File library name specified. The name of the library of the file being referenced, as specified in the source.

Record format level ID. The level ID of the record format referenced.

Record format name. The name of the record format referenced.

Record length. The length of this record is 96.

Record type. The type of this record is '05'.

Reserved. An ignored field.

Message Reference Record

This record is passed when the compiler encounters a reference to a message ID in a message file. For example, a reference is made in DDS source via the MSGCON keyword.

The Message reference record has the following format:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(7)	Message identifier
15	F	CHAR(10)	Message file name specified
25	19	CHAR(10)	Message file library name specified
35	23	CHAR(10)	Message file name used
45	2D	CHAR(10)	Message file library name used
55	37	CHAR(1)	Reserved

Processors for which this record type is applicable: DDS, CMD.

Field Descriptions

Message file library used. The name of the actual library that contains the message file. This may be *CURLIB or *LIBL if the compiler does not resolve to the library name.

Message file library specified. The name of the library that contains the message file, as specified in the source.

Message file name used. The name of the actual message file that was referenced. This name must always be filled in.

Message file name specified. The name of the message file referenced, as specified in the source.

Message identifier. The message ID referenced.

Record length. The length of this record is 56.

Record type. The type of this record is '06'.

Reserved. An ignored field.

External Reference Error Record

This record is passed when processing fails because a referenced object, such as a file or message file, cannot be found. This record does **not apply to includes**.

After passing one or more of these records, the compiler or preprocessor also passes the Abnormal processor end record (see "Abnormal Processor End Record" on page 16-7).

The External reference error record has the following format:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(10)	Object name specified
18	12	CHAR(10)	Object library name specified
28	1C	CHAR(7)	Object type
35	23	CHAR(10)	Object name used
45	2D	CHAR(10)	Object library name used
55	37	CHAR(1)	Reserved

Processors for which this record type is applicable:
SQL/400, DDS, CL, RPG/400, COBOL/400, C/400 processors.

Field Descriptions

Object library name used. The actual name of the library that contains the object that was referenced.

Object library name specified. The name of the library that contains the object that was not found.

Object name used. The actual name of the object that was referenced. This name must always be filled in.

Object name specified. The name of the object referenced that was not found.

Object type. The type of object that was not found.

Record length. The length of this record is 56.

Record type. The type of this record is '15'.

Reserved. An ignored field.

Object Already Exists Error Record

This record is passed when the compiler or preprocessor fails because the object that was to be created exists. There is no REPLACE parameter on the command because the compiler or preprocessor expects the object not to exist.

After passing this record, the compiler or preprocessor must also pass the Abnormal processor end record (see "Abnormal Processor End Record" on page 16-7).

The External reference error record has the following format:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved

Offset		Type	Field
Dec	Hex		
8	8	CHAR(10)	Object name that already exists
18	12	CHAR(10)	Object library name
28	1C	CHAR(7)	Object type
35	23	CHAR(1)	Reserved

Processors for which this record type is applicable: DDS (PF and LF only).

Field Descriptions

Object library name. The name of the library that contains the object that already exists. A specific library name, not *CURLIB or *LIBL must be passed.

Object name that already exists. The name of the object that already exists and could not be replaced.

Object type. The type of the object that already exists.

Record length. The length of this record is 36.

Record type. The type of this record is '16'.

Reserved. An ignored field.

Start of New Program Record

The COBOL/400 compiler has the ability to compile source that contains more than one program. This record is passed by the COBOL/400 compiler when the beginning of a new program is encountered.

The Start of new program record has the following format:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Record length
4	4	CHAR(2)	Record type
6	6	CHAR(2)	Reserved
8	8	CHAR(10)	New program name
18	12	CHAR(10)	Program name created
28	1C	CHAR(10)	Program library name
38	26	CHAR(7)	Message identifier
45	2D	CHAR(3)	Reserved

Processors for which this record type is applicable: COBOL/400 processor.

Field Descriptions

Program library name. The library where the program was created. This field contains blank if an error occurred.

Message identifier. The message ID of the completion message.

New program name. The name of the new program, as per IDENTIFICATION DIVISION.

Program name created. The name of the program created in the previous step. If a program was not created due to syntax errors or because REPLACE(*NO) was specified and the object already existed, this field contains '*ERROR'. If a program was not created because OPTION(*NOGEN) was specified, this field contains '*NONE'.

Record length. The length of this record is 48.

Record type. The type of this record is '40'.

Reserved. An ignored field.

Examples of Records Written

The following examples illustrate how compilers and preprocessors communicate with the Application Development Manager/400 APIs in different circumstances. In all these examples, it is assumed that the compiles are submitted by an Application Development Manager/400 BLDPART command, which means it has called QLYSETS to set the status of the space to *READY before calling the compiler or preprocessor.

It is also assumed that a cleanup is done after the compile by calling QLYSETS again to set the status of the space to *COMPLETE.

Example 1

RPG/400 compiler successfully compiles source that has one include in it.

The compiler first calls QLYGETS and determines that it was started by the BLDPART command. Then it calls QLYWRTBI to pass records of the following record types and in the following order:

1. **Processor start**
2. **Include**
3. **Normal processor end**

Example 2

DDS compiler successfully compiles source of type LF and creates a logical file based on two physical files.

The compiler first calls QLYGETS and determines that it was started by the BLDPART command. Then it calls QLYWRTBI to pass records of the following record types and in the following order:

1. **Processor start**

2. **File reference** (This record is called for the first physical file on which the logical file is based. The based-on indicator is set to Y (yes).
3. **File reference** (This record is called for the second physical file on which the logical file is based. The based-on indicator is set to Y (yes).
4. **Normal processor end**

Example 3

COBOL/400 compiler fails when compiling source that has one include in it because the include was not found in *LIBL.

The compiler first calls QLYGETS and determines that it was started by a BLDPART command. Then it calls QLYWRTBI to pass records of the following record types and in the following order:

1. **Processor start**
2. **Abnormal processor end**

Example 4

COBOL/400 compiler fails when compiling source that references a record format of a database file because the file was not found in *LIBL.

The compiler first calls QLYGETS and determines that it was started by a BLDPART command. Then it calls QLYWRTBI to pass records of the following record types and in the following order:

1. **Processor start**
2. **External reference error** (The name of the *Library specified* passed to QLYWRTBI is *LIBL.)
3. **Abnormal processor end**

Chapter 17. COBOL/400 APIs

This chapter describes the COBOL/400 APIs, which let you control run units and error handling. They are:

- Change COBOL Main Program (QLRCHGCM)
- Retrieve COBOL Error Handler (QLRRTVCE)
- Set COBOL Error Handler (QLRSETCE)

Please refer to "Using COBOL Program to Call APIs" on page A-32 and "Error Handler for Example COBOL Program" on page A-33 for illustrations of how to use these APIs.

Change COBOL Main Program (QLRCHGCM) API

Required Parameter

1	Error code	I/O	Char(*)

The Change COBOL Main Program (QLRCHGCM) API lets you create an additional run unit[1] by assigning a different System/36-compatible, System/38-compatible, or AS/400 COBOL/400 program to serve as a main program. You can call it from any programming language.

After you call this API, the next nonactive COBOL program that runs becomes the main program in a new run unit. An active COBOL program is a program that has been called, and is not in its initial state.

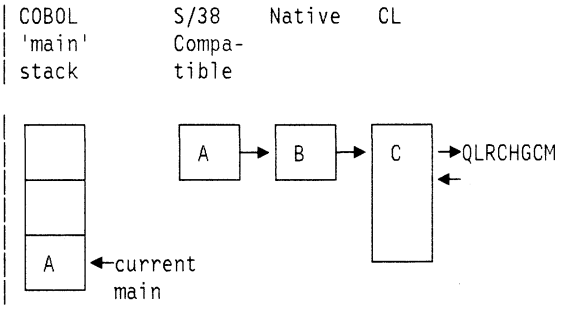
In the following example, System/38-compatible COBOL Program A calls AS/400 COBOL/400 Program B. Because Program A is the first COBOL program, it is the main COBOL program.

COBOL Program B is a menu program that calls CL Program C.

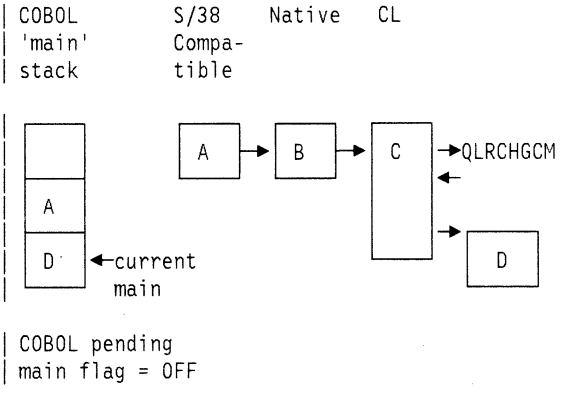
Program C must start a new COBOL application that will pass control back to it, regardless of error conditions. To accomplish this, Program C calls the QLRCHGCM API before calling the new COBOL application.

When program C calls the new COBOL application in the form of Program D, Program D becomes the main program in a new run unit. When Program D's run unit ends, control returns to the original run unit, and Program A becomes the current main program again.

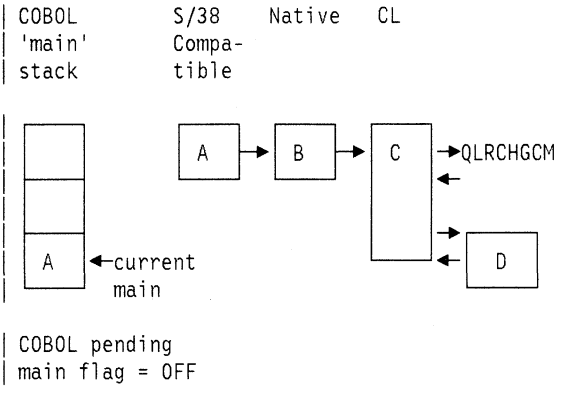
Stage 1



Stage 2



Stage 3



[1] By creating more than one run unit, you can treat files, storage, and error conditions differently than you would using an ordinary subprogram.

Required Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

LBE7040 E Format of error code parameter is not correct.

Retrieve COBOL Error Handler (QLRRTVCE) API

Required Parameter Group

1	Current or pending error-handling exit program name	Output	Char(20)
2	Scope of error-handling exit program	Input	Char(1)
3	Error code	I/O	Char(*)

The Retrieve COBOL Error Handler (QLRRTVCE) API allows you to retrieve the name of the current or pending COBOL error-handling program. You can call it from any programming language.

Required Parameter Group

Current or pending error-handling exit program name
OUTPUT; CHAR(20)
The qualified name of the error-handling program for the current or pending COBOL run unit.

The 20 characters of this parameter are:

1–10 The name of the program object.

Valid values are:
***NONE** No user-defined COBOL error handler has been set.
program-name
The name of the error-handling program.

11–20 The library where the program object existed. The valid value is:
library-name
The library where the program object existed.

Scope of error-handling exit program
INPUT; CHAR(1)
The program can apply to a current or pending run unit.

Valid values are:

C Current COBOL run unit
P Pending COBOL run unit

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

LBE7040 E Format of error code parameter is not correct.
LBE7051 E Scope parameter not valid.
LBE7052 E Run unit specified for error handler does not exist.
LBE7055 E Severe error while addressing parameter list. The API did not complete.

Set COBOL Error Handler (QLRSETCE) API

Required Parameter Group

1	Error-handling exit program name	Input	Char(20)
2	Scope of error-handling program	Input	Char(1)
3	New error-handling exit program library	Output	Char(10)
4	Current or pending error-handling exit program name	Output	Char(20)
5	Error code	I/O	Char(*)

The Set COBOL Error Handler (QLRSETCE) API allows you to specify the identity of a COBOL error-handling program. You can call it from any programming language.

After you call this API, any COBOL/400 program that issues an inquiry message with options C, D, or F will first call the defined error-handling program. This program receives the message identification and substitution text, as well as the name of the program that received it, and a list of valid one-character responses. The defined program is responsible for returning a one-character code (blank, C, D, F, or G) indicating whether the COBOL program should continue or not.

Note: All messages issued by the operating system during the running of a COBOL program are monitored by the COBOL program. Only some of the system messages issued will result in a COBOL inquiry message.

For more information about error handling and the issuing of COBOL inquiry messages, see the chapter on error handling in the *COBOL/400® User's Guide*.

You can define a different error-handling program for each COBOL run unit, but when a new COBOL run unit starts, it uses the error-handling program from the previous run unit.

Only one error-handling program can be active at a time. If an error occurs in the error-handling program, the COBOL program does not call the error-handling program again. (In other words, recursive calls do not occur.) Instead, the inquiry message would be issued as if no error-handling program were defined.

You cannot change the name of the error-handling program while it is responding to an error in a COBOL program.

If an error occurs during the calling of the error-handling program, an informational message (LBE7430) is issued, and processing continues as if no error-handling program were defined.

The error-handling program is defined by the user. The parameters are described under "Error-Handling Exit Program."

Required Parameter Group

Error-handling exit program name
INPUT; CHAR(20)
The qualified name of the error-handling program.

The 20 characters of this parameter are:

1–10 The name of the program object.

Valid values are:
***NONE** No user-defined COBOL error-handling program exists.
program-name
The name of the error-handling program. The name can be an extended one.

11–20 The library where the program object exists.

Valid values are:
***CURLIB**
The current library is used.
***LIBL** The API searches the library list to find the object.
library-name
The name of the library where the program object exists. The name can be an extended one.

Scope of error-handling program
INPUT; CHAR(1)
The program can apply to a current or pending run unit.

Valid values are:

C Current COBOL run unit
P Pending COBOL run unit

New error-handling exit program library
OUTPUT; CHAR(10)
The library where the program object exists. If *CURLIB or *LIBL was specified for the error-handling exit program name parameter, the library returned for this parameter shows the library where the program was found. If *CURLIB or *LIBL was not specified, the library returned here should be the same as character 11 through 20 of the error-handling exit program name parameter. Valid value is:

library-name
The library where the program object exists.

Current or pending error-handling exit program name
OUTPUT; CHAR(20)
The qualified name of the error-handling program that was in place before the current error-handling program was set.

The 20 characters of this parameter are:

1–10 The name of the previous error-handling program object.

Valid values are:
***NONE** No previous current or pending error-handling program existed.
program-name
The name of the error-handling program.

11–20 The library where the previous error-handling program object existed. Valid value is:
library-name
The library where the previous error-handling program object existed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

LBE7040 E Format of error code parameter is not correct.
LBE7050 E Error handler is already responding to an error in the same run unit.
LBE7051 E Scope parameter not valid.
LBE7052 E Run unit specified for error handler does not exist.
LBE7055 E Severe error while addressing parameter list. The API did not complete.
LBE7060 E Error in program name or availability.
LBE7061 E Error in library name or availability.
LBE7062 E Error in library list.

Error-Handling Exit Program

Required Parameter Group

1	COBOL message identification	Input	Char(7)
2	Valid responses to message	Input	Char(6)
3	Name of program issuing error	Input	Char(20)
4	System message causing COBOL message	Input	Char(7)
5	Message text	Input	Char(*)
6	Length of passed message text	Input	Binary(4)
7	Return code	Output	Char(1)

This is a user-defined program that acts as an error handler for a COBOL program. Use the QLRSETCE API to establish this relationship between the two programs.

Required Parameter Group

COBOL message identification
INPUT; CHAR(7)
A 3-character prefix followed by a 4-character number.

Valid responses to message
INPUT; CHAR(6)
A nonnumeric literal containing the list of valid 1-character responses. The list is variable in length and consists of uppercase letters in alphabetical order. The list always ends with a space.

Examples of lists of valid responses:

CG
CDFG

Name of program issuing error
INPUT; CHAR(20)
The qualified name of the COBOL/400 program that issued the error.

The 20 characters of this parameter are:

1—10 The name of the program object. The valid value is:
program-name
The name of the program object.

11—20 The library where the program object existed. The valid value is:
library-name
The library where the program object existed.

System message causing COBOL message
INPUT; CHAR(7)
Some COBOL error messages are issued because of error messages received from the system. This parameter identifies such system messages.

Valid values are:

***NONE** No system message is available.
message-id
A 3-character message prefix followed by a 4-character number.

Message text
INPUT; CHAR(*)
The substitution text of the message, its length determined by Parameter 6.

Length of passed message text
INPUT; Binary(31)
If the original message was a system message, the substitution text for the system message is passed. In the absence of an original system message, Parameter 4 has a value of *NONE, and the substitution text for the COBOL message is passed.

Return code
OUTPUT; CHAR(1)
Must be one of the values specified in Parameter 2, or a space. If the value is not one of these, a response of a space is assumed.

Valid values are:

blank Issue the COBOL message that was passed to the error-handling program.
G Continue running the COBOL program.
C End the current COBOL run unit.
D Same as C, but produce a formatted dump of user-defined COBOL variables.
F Same as D, but also dump COBOL's internal variables.

Part 7. Message Handling APIs

Chapter 18. Message Handling APIs

The message handling APIs let your applications work with AS/400 messages. You can use these APIs to send messages to various destinations, sharing status and error information between programs only or between programs and users.

Before using the message handling APIs, read the following material:

- "Terms and Concepts." This section briefly describes the elements of AS/400 messages and message handling. For complete background information, see the chapters about defining and working with messages in the *CL Programmer's Guide*.
- "Error Handling in the Extended Program Model (EPM) Environment" on page 18-3. This section explains how to use the APIs in extended program model (EPM) programs.

The message handling APIs are presented in alphabetical order beginning on page 18-4. "Diagnostic Reporting" on page A-19 shows how to use the APIs to send and receive messages.

The message handling APIs include the following:

Move Program Messages (QMHMOVPM) moves messages from one program's message queue to the message queue of a program higher up the program stack. This is especially useful for error handling.

Receive Nonprogram Message (QMHRCVM) receives a message from a nonprogram message queue, providing information about the sender of the message as well as the message itself. This API is similar in function to the Receive Message (RCVMSG) command with the MSGQ parameter.

Receive Program Message (QMHRCVPM) receives a message from a program message queue, providing information about the sender of the message as well as the message itself. This API is similar in function to the Receive Message (RCVMSG) command with the PGMQ parameter.

Remove Nonprogram Messages (QMHRMVM) removes messages from nonprogram message queues. After you remove a message, you can no longer work with that instance of the message in any way. This API is similar in function to the Remove Message (RMVMSG) command with the MSGQ parameter.

Remove Program Messages (QMHRMVPM) removes messages from program message queues. After you remove a message, you can no longer work with that instance of the message in any way. This API is similar in function to the Remove Message (RMVMSG) command with the PGMQ parameter.

Resend Escape Message (QMHRSNEM) resends an escape message from one program's message queue to the message queue of the previous program in the program stack.

Retrieve Message (QMHRTVM) Retrieves the message text and other elements of a predefined message stored in a message file on your AS/400 system. This API is similar to the Retrieve Message (RTVMSG) command.

Retrieve Request Message (QMHRTVRQ) retrieves request messages from the current job's job message queue.

Send Break Message (QMHSNDBM) sends a message to a work station for immediate display, interrupting the work station user's task. You can use break messages to warn users of impending system outages and the like. This API is similar in function to the Send Break Message (SNDBRKMSG) command.

Send Nonprogram Message (QMHSNDM) sends a message to a system user or a message queue that is not associated with a specific program. This API is similar in function to the Send Program Message (SNDPGMMSG) command with the TOMSGQ parameter.

Send Program Message (QMHSNDPM) sends a message to the message queue of a program in the program stack. This API is similar in function to the Send Program Message (SNDPGMMSG) command with the TOPGMQ parameter.

Send Reply Message (QMHSNDRM) sends a response to an inquiry message. This API is similar in function to the Send Reply (SNDRPY) command.

Chapter 19, "Network Management APIs," describes three related APIs that you can use to create, retrieve, and send alertable messages:

Generate Alert (QALGENA)
Retrieve Alert (QALRTVA)
Send Alert (QALSNDA)

Chapter 22, "Operational Assistant APIs," describes the Operational Assistant* APIs you can use to work with messages:

Send Message (QEZSNDMG)
Work with Messages (QEZMSG)

Terms and Concepts

This section describes the basic elements of AS/400 messages and message handling, including:

- Immediate and predefined messages
- Message types, or the purposes assigned to messages when they are sent
- Message destinations, such as work station displays and program message queues

For details about these topics, see the *CL Programmer's Guide*.

Immediate and Predefined Messages

The OS/400 program uses two basic kinds of messages, immediate and predefined. **Immediate messages**, also known as **impromptu messages**, are created by the sender when they are sent. They consist completely of text that the sender supplies. They do not have identifying codes like CPF2469, and they are not stored in message files on the system.

In contrast, **predefined messages** are created and exist outside the programs that use them. Each predefined message has its own **message definition**, which is stored in a message file (object type *MSGF) on the system. The message definition includes the message text and other items, such as message help, the default reply for the message, and its severity level.

Each message definition has its own message identifier. The **message identifier** is a 7-character code, such as CPF2469, that refers to the message definition as a whole. It is displayed with the text of the message and lets programs refer to the message easily. The term message identifier is usually abbreviated to MSGID in items such as CL commands.

Predefined messages have two types of text, message text and message help. The **message text**, also known as **first-level text**, is the version of the predefined message that is first displayed at a work station or logged in a job log. It briefly describes a condition. **Message help**, also known as **second-level text**, is available to the user on request. It usually describes the cause of the condition and details any corrective action the user should take.

The message text and message help for predefined messages can consist of only constant text, or of constant text and substitution variables. **Substitution variables** are variables for items such as file names in the message text or message help. They allow the message to better describe the recipient's individual circumstances. The specific values you assign to these variables are called **message data**.

When either an immediate or predefined message is sent, the command or API used to send it usually assigns it a message key. The **message key**, also called the **message reference key**, is a unique string of characters that identifies a particular instance of a message in a queue. (In contrast, a message identifier identifies a message as it is stored in a message file, not as it is sent.) You can use the message key to refer to a specific instance of a message in order to move, receive, reply to, resend, or remove it.

Message Types

The OS/400 program divides messages into types according to their use. These **message types** are categories based on the message's purpose. The sender of the message determines its type when sending the message. The message handling APIs use these message types in defining and working with messages:

Completion (*COMP). Reports the successful completion of a task.

Diagnostic (*DIAG). Describes errors in processes or input data. When an error occurs, a program usually sends an escape message, which causes the task to end abnormally. One or more diagnostic messages can be sent before the escape message to describe the error.

Escape (*ESCAPE). Indicates a condition causing a program to end abnormally, without completing its work.

Exception (*EXCP). Indicates a condition causing a program to end abnormally, without completing its work. An exception message can be either an escape or a notify message. The exception message type and the special value *EXCP are used only with the Receive Program Message (QMHRCVPM) API.

Informational (*INFO). Conveys information *without* asking for a reply.

Inquiry (*INQ). Conveys information *and* asks for a reply.

Notify (*NOTIFY). Describes a condition requiring corrective action or a reply from the calling program.

Reply (*RPY). Responds to an inquiry or notify message.

Request (*RQS). Requests a function from the receiving program.

Sender's copy (*COPY). Is a copy of an inquiry or notify message. This copy is kept by the sender of the inquiry or notify message.

Status (*STATUS). Describes the status of work being done by a program.

Message Destinations

All AS/400 messages are sent to message queues. The system provides the following message queues:

- A work station message queue for each work station on the system.
- A user profile message queue for each enrolled user.
- A job message queue for each job running on the system. The job message queue consists of two or more message queues:
  1. An **external message queue (*EXT)** to send messages between an interactive job and the work station user.
  2. A set of **program message queues** corresponding to the programs called in the job. Each call of a program within a job has a program message queue. If a program appears in the program stack twice, it has two separate program message queues.

  All message queues other than external and program message queues are considered **nonprogram message queues.**
- The system operator's message queue, QSYSOPR.
- The history log, QHST.

A message sent to a message queue remains on the queue until you use a command or API to remove it from the queue or move it to another queue. **Removing a message** from a message queue deletes that one instance of the message from the system. After removal, you can no longer receive or otherwise work with that instance of

the message. However, other instances of the message might still be available in the same or different message queues, and the message definition of a predefined message still exists in a message file.

Error Handling in the Extended Program Model (EPM) Environment

This section explains how to use the message handling APIs in conjunction with error handling routines, defined by the user, in programs running in the extended program model (EPM) environment.

The **extended program model** (EPM) works with the system to provide functions such as a common run-time environment, debugging tools, application library routines, an exception handler, and a session manager. For a detailed description of the EPM, see the *C/400® User's Guide* or the *Pascal User's Guide*.

Programs written in languages that use the EPM are called **EPM programs.** The following lists show which AS/400 programming languages use the EPM:

EPM Languages	**Non-EPM Languages**
C/400	BASIC
FORTRAN/400	COBOL/400
Pascal	Control Language (CL)
	Machine interface (MI)
	PL/I
	REXX
	RM/COBOL
	RPG
	System C/400 PRPQ

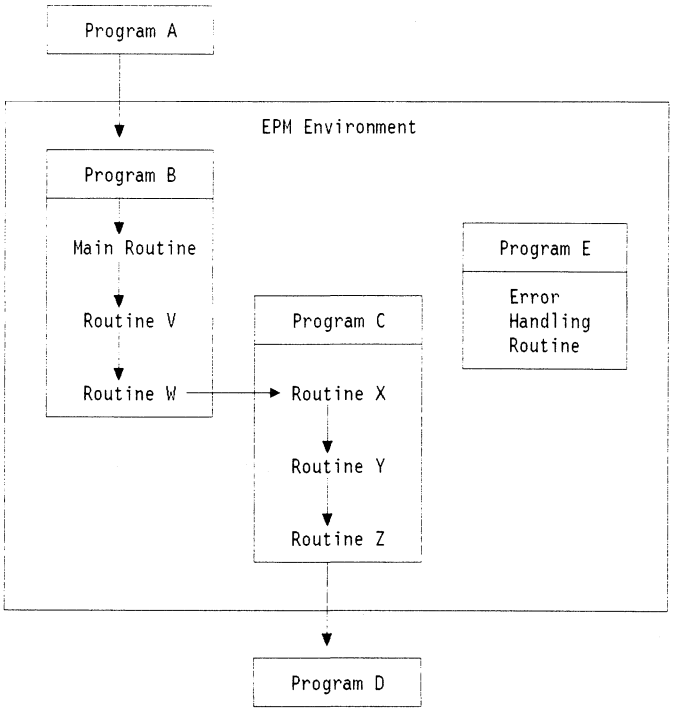
The EPM exception handler can work with the message handling APIs to respond to exceptions that occur in EPM programs running on an AS/400 system. When an error occurs, you can:

- Use the Send Program Message (QMHSNDPM) API to send an escape message from your EPM program's error handling routine to a non-EPM program.
- Use existing EPM functions to resend an escape message from your EPM program's error handling routine to an EPM or non-EPM program.
- Return control from your EPM program's error handling routine to the routine or program that was active before the error occurred, and use one or more message handling APIs to handle the error.

These capabilities are described in the following sections. The first section describes an EPM programming scenario. The sections after that use the scenario as a basis for examples of how to use the EPM capabilities and the APIs together to handle errors.

Example Scenario

This diagram depicts a scenario used by the examples in the following sections.



The diagram shows five programs:

Program A Not an EPM program. Program A is written in CL, RPG, COBOL, or PL/I, not the C/400 or Pascal language.

Program B An EPM program. Program B is a separately compiled C/400 or Pascal program containing three routines named Main, V, and W.

Program C An EPM program. Program C is a separately compiled C/400 or Pascal program containing three routines named X, Y, and Z.

Program D Not an EPM program. Program D is written in CL, RPG, COBOL, or PL/I, not the C/400 or Pascal language.

Program E An EPM program. Program E is written in the C/400 or Pascal language and contains a user-defined error handling routine for the EPM environment. In a Pascal program, the routine must be named ONERROR. In a C/400 program, the routine can have any name; you define the name with the SIGNAL function.

The programs perform as follows:

1. Program A calls program B, which implicitly creates an EPM environment.
2. The main routine in program B begins running. If an error occurs and the program is written in the Pascal language, the ONERROR routine is automatically called. If an error occurs and the program is written in the C/400 language, the main routine uses the SIGNAL function to direct any errors to the error handling routine in program E.
3. Program B's main routine calls routine V, and routine V calls routine W. These are internal calls, so the Display Active Programs display does not list the

main routine or routines V and W; it lists only programs A and B. The main routine and routines V and W are on a stack maintained by the EPM. To view the stack, use the EPM debugger command DISPLAY TRACE.

4. Routine W calls routine X. This is an external call to program C.
5. Routine X calls routine Y, which calls routine Z. These are both internal calls.
6. Routine Z calls program D; this is an external call to a non-EPM program.
7. Program D calls the Send Program Message (QMHSNDPM) API to send an escape message to program C.
8. The EPM exception handler is called automatically. It passes control to the error handling routine in program E.
9. The error handling routine in program E begins running.

Example: Send a New Message from the Error Handling Routine

You can call the QMHSNDPM API from the error handling routine in program E to send escape messages to non-EPM programs. For example, you can call the QMHSNDPM API to send a new escape message to program A. This causes the EPM environment to end.

You cannot call the QMHSNDPM API from the error handling routine to send an escape message to another program in the EPM environment. For example, you cannot use the QMHSNDPM API to send an escape message to program B. You can send an escape message from one EPM program to another, but not from within the error handling routine; see "Example: Return to Routine Z."

Example: Resend an Escape Message

You can use existing EPM error handling functions to resend escape messages in the EPM environment. You do not need the message handling APIs for this task.

To resend the escape message to routine Y, the previous EPM routine, set XRESIGPRIOR in the FACTION set and return from your error handling routine. EPM support resends the escape message to routine Y. Your error handling routine is called again but this time from routine Y.

To resend the escape message to program A and end the EPM environment, set XRESIGOUTER in the FACTION set and return from your error handling routine. EPM support resends the escape message to program A (the program that called the main EPM routine), and all EPM programs are removed from the program stack.

You cannot call the Resend Escape Message (QMHRSNEM) API from within an error handling routine in an EPM program. This is because of the way exceptions are handled in the EPM environment; see the EPM information in the *C/400* User's Guide* or the *Pascal User's Guide* for details.

Example: Return to Routine Z

Instead of using the techniques in the previous examples, you can handle the error in routine Z. In your error handling routine in program E, set a flag to indicate that an exception occurred, and return to routine Z. Routine Z resumes running at the instruction after the call to program D. Routine Z can then test the flag to determine whether an exception occurred. At that point, routine Z can either continue processing or call one of these APIs:

- Move Program Messages (QMHMOVPM) to move messages to the program message queue for program A or program B.
- Resend Escape Message (QMHRSNEM) to resend the escape message to program B. (Routine W takes control because it is the last routine called in program B.) The EPM exception handler then calls the error handling routine that is active in program B.
- Send Program Message (QMHSNDPM) to send a new escape message to any program in the program stack (in this case, program A or program B). If you send the escape message to program A, the EPM environment ends. If you send it to program B, the EPM exception handler takes control and calls the error handling routine that is active in program B.

Move Program Messages (QMHMOVPM) API

Parameters

Required Parameter Group:

1	Message key	Input	Char(4)
2	Message types	Input	Array of Char(10)
3	Number of message types	Input	Binary(4)
4	Program message queue	Input	Char(10)
5	Program stack counter	Input	Binary(4)
6	Error code	I/O	Char(*)

The Move Program Message (QMHMOVPM) API moves one or more messages from the current program's message queue to a previous program's message queue. Moving a message does not change the sender information stored with the message. However, moving an escape message automatically changes it to a diagnostic message.

You can use the QMHMOVPM API to pass messages up the program stack, transferring important information to a previous program. When messages are sent to a program and that program ends without moving its (the ending program's) messages up the stack, none of the programs left in the stack can receive those messages.

Assume a program is sent several diagnostic messages and one escape message in response to an error. Another program up the stack can handle the error, so the first program uses the QMHMOVPM API to move all the diagnostic messages to the program up the stack. It then uses

"Resend Escape Message (QMHRSNEM) API" on page 18-16 to send the escape message to the calling program.

Required Parameter Group

Message key
INPUT; CHAR(4)
When moving a single, specific message, the key to that message. The key is assigned by the command or API that sends the message.

When using the message-types parameter to move a group of messages, use blanks for this parameter.

Message types
INPUT; ARRAY of CHAR(10)
The type or types of the messages being moved.

When moving a single message by specifying the message key in the preceding parameter, use blanks for this parameter.

When moving a group of messages, specify a list of one through four message types. You can use these values in the list to move all messages of one or more types:

***COMP**	Completion
***DIAG**	Diagnostic
***ESCAPE**	Escape. After an escape message is moved, its message type changes to diagnostic.
***INFO**	Informational

For descriptions of the message types, see "Message Types" on page 18-2.

If there are no messages of a type you specify, the QMHMOVPM API does not return an error. It simply returns control to the calling program.

If the number-of-message-types parameter specifies 0, this parameter is ignored.

Number of message types
INPUT; BINARY(4)
The number of message types specified in the message-types parameter.

If you use blanks for the message-key parameter and specify one or more message types, the value must be 1 through 4.

If you specify a message key and use blanks for the message-types parameter, the value must be 0.

Program message queue
INPUT; CHAR(10)
The name of the program message queue to move the messages to, or the name of the program to start counting from when using a value other than 0 for the program-stack-counter parameter. The program you specify must be in the program stack, and you cannot specify the external message queue.

You can specify a program by name or use this special value:

- ***** The message queue of the current program (that is, the program moving the messages).

Program stack counter
INPUT; BINARY(4)
A number identifying the location in the program stack of the program message queue to which messages are being moved. The number is relative to the program specified in the program-message-queue parameter. It indicates how many calls up the program stack the target program message queue is from the current one specified in the program-message-queue parameter. Valid values are:

- **0** Move the messages to the message queue of the program specified in the program-message-queue parameter. You cannot use 0 when the program-message-queue parameter specifies * to designate the current program's message queue.
- **1** Move the messages to the queue of the program that calls the program specified in the program-message-queue parameter.
- **n (any positive number)** Move the messages to the queue of the nth program up the stack from the program specified in the program-message-queue parameter.

  You can use any positive number that does not exceed the actual number of programs in the program stack. Do not include the external message queue in your count.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24A3	E	Value for program stack counter not valid.
CPF24A5	E	Value of &1, for number of message types, not valid.
CPF24B3	E	Message type &1 not valid.
CPF24B4	E	Severe error while addressing parameter list.
CPF24B5	E	Not able to move message.
CPF2410	E	Message key not found in message queue &1.
CPF2460	E	Message queue &1 could not be extended.
CPF2477	E	Message queue &1 currently in use.
CPF2479	E	Program message queue &1 not found.
CPF2508	E	Move of message to same program message queue not valid.
CPF2509	E	Message reference key &2 points to message not belonging to mover's program message queue.
CPF3CF1	E	Error code parameter not valid.

Receive Nonprogram Message (QMHRCVM) API

Parameters

Required Parameter Group:

1	Message information	Output	Char(*)
2	Length of message information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified message queue name	Input	Char(20)
5	Message type	Input	Char(10)
6	Message key	Input	Char(4)
7	Wait type	Input	Binary(4)
8	Message action	Input	Char(10)
9	Error code	I/O	Char(*)

The Receive Nonprogram Message (QMHRCVM) API receives a message from a nonprogram message queue. To receive a message from a program message queue or from the external message queue, see "Receive Program Message (QMHRCVPM) API" on page 18-11.

Authorities and Locks

Message File Authority	*USE
Message File Library Authority	*USE
Message Queue Authority	*CHANGE if the message-action parameter specifies *REMOVE; *USE for other message actions

Required Parameter Group

Message information
OUTPUT; CHAR(*)
The variable that receives the information returned, in the format specified in the format-name parameter, of the length specified in the length-of-message-information parameter.

Length of message information
INPUT; BINARY(4)
The size of the area to contain the message information, in bytes. The minimum size is 8.

This parameter must specify the size of the variable you use for the message-information parameter. If this parameter specifies a longer size, other parts of storage could be overwritten when the API returns the information.

The API returns as much information as it can fit in this length. If the available message information is longer, it is truncated. If the available message information is shorter, the unused output area is unchanged; whatever is already stored in that space remains there.

To determine how much information the API actually returns in response to this call, see the bytes-returned field in the RCVM0100 or RCVM0200 output. To determine how much information the API could return if space were available, see the bytes-available field.

Format name
INPUT; CHAR(8)
The format to use for the message information. Specify one of these format names:

RCVM0100 Brief message information. For details, see "RCVM0100 Format" on page 18-8.
RCVM0200 All message information. For details, see "RCVM0200 Format" on page 18-8.

Qualified message queue name
INPUT; CHAR(20)
The name of the message queue from which to receive the message, and the library in which it resides. The first 10 characters specify the message queue, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

You cannot receive messages from the history log, QHST.

Message type
INPUT; CHAR(10)
The type of the message being received. The message-type and message-key parameters work together. Depending on the message type, the key can be required, optional, or disallowed. For a list of valid message types and information about how they work with the message-key parameter, see "Message Types and Message Keys" on page 18-7.

Message key
INPUT; CHAR(4)
The key to the message being received. The key is assigned by the command or API that sends the message.

What you can use for this parameter depends on what you use for the message type. For details, see "Message Types and Message Keys" on page 18-7.

If you are not receiving messages by key, use blanks for this parameter.

If you specify a key and the message queue does not contain a message with that key, an error is returned.

If you specify a key and the message queue does contain a message with that key, the API might or might not return a message; it never returns an error in this case. Whether or not the API returns a message depends on the value of the message-type parameter. For example, if you specify the message type *PRV (previous) and the queue does not contain a message before the message with the key, the API does not return a message. Because the key you specified is valid, the API does not return an error either.

You can receive the reply to an inquiry message through the key to the sender's copy of the inquiry. If the reply is not available, no message is returned, and the API does not return an error.

When the message type is the special value *NEXT, you can use the special value *TOP for the message key. *TOP returns the message at the top of the queue.

When the message type is the special value *NEXT or *PRV (previous), you can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation.

Wait time
INPUT; BINARY(4)
The length of time to wait for the message to arrive in the queue so it can be received.

The system ignores this parameter when you specify both a message key and a message type other than reply (*RPY). The parameter is used in only two cases:

1. The message type is reply (*RPY), and the message-key parameter specifies the key to the sender's copy of the message.
2. The message type is anything except reply (*RPY), and the message-key parameter is blank. In this case, the QMHRCVM API does not use the wait-time parameter immediately. First, the API checks the queue for the first message of that type that has not been received. If no such message is found, the API then waits the specified length of time for a message to arrive.

Valid values are:

0 Do not wait for the message. You must use 0 if you specify a message key and the message is not a reply message.
−1 Wait until the message arrives in the queue and is received, no matter how long it takes. The system has no limit for the wait time.
n (any positive number)
Wait **n** seconds for the message to arrive in the queue.

If you specify a value of zero or above and the message does not arrive in the queue in the specified time, most fields in the RCVM0100 or RCVM0200 output returned by the API are unchanged. The bytes-returned output field has a value of 8, and the bytes-available output field has a value of 0. The remaining output fields are unchanged; they contain whatever was already stored in the space.

Message action
INPUT; CHAR(10)
The action to take after the message is received. Valid values are:

***OLD** Keep the message in the message queue and mark it as an old message. You can receive the message again only by using the message key or by specifying the message type *NEXT, *PRV (previous), *FIRST, or *LAST.

***REMOVE** Remove the message from the message queue. The message key is no longer valid, so you cannot receive the message again.
***SAME** Keep the message in the message queue without changing its new or old designation. *SAME lets you receive the message again later without using the message key.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Message Types and Message Keys

The message-type and message-key parameters work together. Depending on the message type, the key can be required, optional, or disallowed. The following table lists each message type you can specify, tells whether or not it requires a key, and gives other information about using that type.

When used without a key, most message types receive only new messages. When used with a key, most can receive old or new messages. Message types *FIRST, *LAST, *PRV (previous), and *NEXT always receive both old and new messages.

New messages are messages that have been sent to a queue and have not yet been received. **Old messages** are messages that have been received but have not yet been removed from the queue.

All message types listed in the following table are received in first-in first-out (FIFO) order.

The following terms are used to describe the message key in the following table:

Disallowed Do not specify a message key. Instead, use blanks for the message-key parameter. Specifying a message key results in an error.
Required Specify a message key. Not specifying a message key results in an error.
Optional You can either specify a message key or use blanks for the message-key parameter.

When you do not specify a message key, the first new message of the specified type is received. If a new message of that type is not in the message queue, no error is returned, and the unused space allowed for the output in the message-information parameter is unchanged.

When you specify a message key and the message in the message queue is of the type specified, the message is received. If the message is not found, or if the message found does not match the type specified, an error code or exception is returned.

There are two cases where the message is not found and no error is returned. In both cases, the bytes returned

field equals 8 and the bytes available field equals 0. The two cases are:

- Receiving without a message key (the key is optional or disallowed). A message of the specified type is not found in the queue.
- Receiving with a message key (the key is required) and the message type is *PRV or *NEXT. The message with the key specified was found in the queue, but no *PRV or *NEXT message is found.

The message types you can specify in the QMHRCVM API follow:

Message Type	Message Key	Description
*ANY	Optional	Receives a message of any type except sender's copy.
*COMP	Optional	Receives a completion message.
*COPY	Required	Receives a copy of a previously sent inquiry message. The qualified-message-queue-name parameter must specify the reply message queue specified when the inquiry was sent.
*DIAG	Optional	Receives a diagnostic message.
*FIRST	Disallowed	Receives the first new or old message in the queue.
*INFO	Optional	Receives an informational message.
*INQ	Optional	Receives an inquiry message. If the message action is *REMOVE and a reply to the inquiry message has not been sent yet, the default reply is automatically sent when the inquiry message is received.
*LAST	Disallowed	Receives the last new or old message in the queue.
*NEXT	Required	Receives the next new or old message after the message with the specified key. You can use the special value *TOP for the message key. *TOP designates the message at the top of the message queue. You can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation.
*PRV	Required	Receives the new or old message before the message with the specified key. You can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation.
*RPY	Optional	Receives the reply to an inquiry message. For the message key, you can use the key to the sender's copy of the inquiry or notify message.

RCVM0100 Format

The following table lists the fields returned in the RCVM0100 format. For more information about each item of information, see "Field Descriptions" on page 18-9.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	CHAR(7)	Message identifier
19	13	CHAR(2)	Message type
21	15	CHAR(4)	Message key
25	19	CHAR(15)	Reserved
40	28	BINARY(4)	Length of message data or text returned
44	2C	BINARY(4)	Length of message data or text available
48	30	CHAR(*)	Message data or text

RCVM0200 Format

The following table lists the fields returned in the RCVM0200 format. For more information about each item of information, see "Field Descriptions" on page 18-9.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	CHAR(7)	Message identifier
19	13	CHAR(2)	Message type
21	15	CHAR(4)	Message key
25	19	CHAR(10)	Message file name
35	23	CHAR(10)	Message file library specified
45	2D	CHAR(10)	Message file library used
55	37	CHAR(10)	Sending job
65	41	CHAR(10)	Sending user profile
75	4B	CHAR(6)	Sending job's number
81	51	CHAR(12)	Sending program

Offset		Type	Field
Dec	Hex		
93	5D	CHAR(4)	Sending program's instruction number
97	61	CHAR(7)	Date sent
104	68	CHAR(6)	Time sent
110	6E	CHAR(10)	Receiving program
120	78	CHAR(4)	Receiving program's instruction number
124	7C	CHAR(11)	Reserved
135	87	CHAR(9)	Alert option
144	90	CHAR(8)	Reserved
152	98	BINARY(4)	Length of message data or text returned
156	9C	BINARY(4)	Length of message data or text available
160	A0	BINARY(4)	Length of message returned
164	A4	BINARY(4)	Length of message available
168	A8	BINARY(4)	Length of message help returned
172	AC	BINARY(4)	Length of message help available
176	B0	CHAR(*)	Message data or text
The offsets to these fields equal the offset to the last fixed-length field plus the length of the previous variable-length fields.		CHAR(*)	Message
		CHAR(*)	Message help

Field Descriptions

The following field descriptions apply only when a message is received. If no message is found, only the bytes-available and bytes-returned fields contain new values. The remaining fields contain whatever information was already stored in the space allowed for the output.

Alert option. Whether and when an SNA alert is created and sent for the message. If a message is received, the value is one of the following:

**DEFER*	An alert is sent after local problem analysis.
**IMMED*	An alert is sent immediately, and the message is sent to the QHST message log at the same time.
**NO*	No alert is sent.
**UNATTEND*	An alert is sent immediately when the system is running in unattended mode (that is, when the value of the alert status network attribute, ALRSTS, is *UNATTEND).

For more information, see the *Alerts and DSNX Guide*.

Bytes available. The length of all available information that could be returned for the format. Bytes available can be greater than the length specified in the API's length-of-message-information parameter. If it is greater, the information returned in the message-information parameter is truncated to the length specified.

Bytes returned. The length of all information returned in the format. The value of the bytes-returned field is always less than or equal to the length of the message-information parameter, and less than or equal to the bytes available. There is one exception to this: When you attempt to receive a message and the message is not found, the value of the bytes-returned field is 8, the value of the bytes-available field is 0, and the remaining fields are unchanged (that is, they contain whatever was already stored in that space). If the bytes-returned value is less than the length specified in the length-of-message-information parameter, the extra space in the message-information parameter is unchanged.

Date sent. The date on which the message was sent, in CYYMMDD (century, year, month, day) format.

Length (general information about the following length fields). These formats use two types of length fields, each related to a single variable-length text field. (The variable-length text fields return information to the caller.) The first type of length field is returned length; the second is available length. **Returned length** is the actual length of the text in the variable-length text field. **Available length** is the length of the text before it is placed in the variable-length text field. It is always greater than or equal to the returned length. If the available length equals the returned length, all the message information is returned. If the text is truncated when placed in the variable-length field, the available length is greater than the returned length by the number of characters truncated.

Length of message available. The length of the available message text, in bytes. If an immediate message is received, the value of this field is zero.

Length of message data or text available. The length of the available immediate message text or message data, in bytes. If the message identifier is not blank, this field contains the length of the available message data for a predefined message. If the message identifier is blank, this field contains the length of the available text of an immediate message.

Length of message data or text returned. The length of the returned immediate message text or message data, in bytes. If the message identifier is not blank, this field contains the length of the message data. If the message identifier is blank, this field contains the length of the immediate message text.

Length of message help available. The length of the available message help text, in bytes. If an immediate message is received, the value of this field is zero.

Length of message help returned. The length of the returned message help text, in bytes. If an immediate message is received, the value of this field is zero.

Length of message returned. The length of the returned text of a predefined message, in bytes. If an immediate message is received, the value of this field is zero.

Message. The text of a predefined message. If an immediate message is received, this field is blank.

The API can truncate the message to fit the available space. If truncation occurs in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters and ends the data with a DBCS shift-in character.

Message data or text. The values for substitution variables in a predefined message, or the text of an immediate message. If the message identifier is not blank, this field contains message data. If the message identifier is blank, this field contains immediate message text.

If this field contains message data and that data contains pointer data, each pointer must start on a 16-byte boundary.

The API can truncate the data or text to fit the available space. If the field contains the text of an immediate message and it is truncated in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters and ends the data with a DBCS shift-in character. However, if the field contains data for a predefined message, the API does not check for DBCS data. This is because message data can contain pointers, and pointers can contain the same characters used to mark DBCS data.

Message file. The name of the message file containing the message received.

Message file library specified. The name of the library containing the message file, as specified in the call to this API. If you specify *CURLIB or *LIBL for the library when you send the message, that value is returned as the library here. For the actual library used when the message is sent, see the message-file-library-used field.

Message file library used. The name of the library used to send the message. Because the library can contain override instructions, this is not necessarily the library in which the message actually resides.

Message help. The message help for the message received. If an immediate message is received, this field is blank.

The API can truncate the message help to fit the available space. If truncation occurs in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters and ends the data with a DBCS shift-in character.

Message identifier. The identifying code of the message received. If an immediate message is received, this field is blank.

Message key. The key to the message received. The key is assigned by the command or API that sends the message. If the message-action parameter specifies *REMOVE, this field is blank.

Message severity. The severity of the message received. Possible values are 0 through 99.

Message type. The message type of the message received. The possible values and their meanings are:

Value	Message Type
01	Completion
02	Diagnostic
04	Informational
05	Inquiry
06	Sender's copy
08	Request
10	Request with prompting
14	Notify
15	Escape
21	Reply, not validity checked
22	Reply, validity checked
23	Reply, message default used
24	Reply, system default used
25	Reply, from system reply list

Receiving program. The name of the program to which the message being received was sent.

Receiving program's instruction number. The number of the last program instruction run before the message being received was sent.

Reserved. An ignored field.

Sending job. The name of the job in which the message being received was sent.

Sending job's number. The job number of the job in which the message being received was sent.

Sending program. The name of the program used to send the message being received.

Sending program's instruction number. The number of the program instruction that issued the command or called the API used to send the message being received.

Sending user profile. The name of the user profile that sent the message being received.

Time sent. The time at which the message being received was sent, in HHMMSS (hour, minute, second) format.

Error Messages

CPF24AF E Message key not allowed with message type specified.
CPF24A7 E Value for the length of message information not valid.
CPF24A8 E Value for wait time not valid.
CPF24A9 E Value for message action not valid.

CPF24B1 E	Message key required for message type specified.
CPF24B2 E	Message key of *TOP requires message type of *NEXT.
CPF24B3 E	Message type &1 not valid.
CPF24B4 E	Severe error while addressing parameter list.
CPF2401 E	Not authorized to library &1.
CPF2403 E	Message queue &1 in &2 not found.
CPF2407 E	Message file &1 in &2 not found.
CPF2408 E	Not authorized to message queue &1.
CPF2410 E	Message key not found in message queue &1.
CPF2411 E	Not authorized to message file &1 in &2.
CPF2433 E	Function not allowed for system log message queue &1.
CPF2450 E	Work station message queue &1 not allocated to job.
CPF2451 E	Message queue &1 is allocated to another job.
CPF2477 E	Message queue &1 currently in use.
CPF2531 E	Message file &1 in &2 damaged for &3.
CPF2548 E	Damage to message file &1 in &2.
CPF2551 E	Message key and message type combination not valid.
CPF3CF1 E	Error code parameter not valid.
CPF3C21 E	Format name &1 is not valid.
CPF8100 E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9830 E	Cannot assign library &1.

Receive Program Message (QMHRCVPM) API

Parameters

Required Parameter Group:

1	Message information	Output	Char(*)
2	Length of message information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Program message queue	Input	Char(20)
5	Program stack counter	Input	Binary(4)
6	Message type	Input	Char(10)
7	Message key	Input	Char(4)
8	Wait time	Input	Binary(4)
9	Message action	Input	Char(10)
10	Error code	I/O	Char(*)

The Receive Program Message (QMHRCVPM) API receives a message from a program or external message queue and returns information describing the message to the calling program.

To receive a message from a message queue other than a program or external message queue, see "Receive Non-program Message (QMHRCVM) API" on page 18-5.

For an example program using the QMHRCVPM API, see "Diagnostic Reporting" on page A-19.

Authorities and Locks

Message File Authority *USE
Message File Library Authority *USE

Required Parameter Group

Message information
OUTPUT; CHAR(*)
The variable that receives information returned, in the format specified in the format-name parameter, of the length specified in the length-of-message-information parameter.

Length of message information
INPUT; BINARY(4)
The size of the area to contain the message information, in bytes. The minimum size is 8.

This parameter must specify the size of the variable you use for the message-information parameter. If this parameter specifies a longer size, other parts of storage could be overwritten when the API returns the information.

The API returns as much information as it can fit in this length. If the available message information is longer, it is truncated. If the available message information is shorter, the unused output area is unchanged; whatever is already stored in that space remains there.

To determine how much information the API actually returns in response to this call, see the bytes-returned field in the RCVM0100 or RCVM0200 output. To determine how much information the API could return if space were available, see the bytes-available field.

Format name
INPUT; CHAR(8)
The format to use for the message information. Specify one of these format names:

RCVM0100 Brief message information. For details, see "RCVM0100 Format" on page 18-8.
RCVM0200 All message information. For details, see "RCVM0200 Format" on page 18-8.

Program message queue
INPUT; CHAR(10)
The name of the program message queue from which to receive messages, or the name of the program to start counting from when using a value other than 0 for the program-stack-counter parameter. The program you specify must be in the program stack. You can specify a program by name or use one of these special values:

* The message queue of the current program (that is, the program receiving the message).
***EXT** The external message queue. The program-stack-counter parameter is ignored. You cannot receive escape messages from this queue.

Program stack counter
INPUT; BINARY(4)
A number identifying the location in the program stack of the program message queue from which messages are received. The number is relative to the program specified in the program-message-queue parameter.

It indicates how many calls up the program stack the originating program message queue is from the one specified in the program-message-queue parameter. Valid values and their meanings are:

0 Receive the message from the queue of the program specified in the program-message-queue parameter.

1 Receive the message from the queue of the program that calls the program specified in the program-message-queue parameter.

n (any positive number)
Receive the message from the queue of the nth program up the stack from the program specified in the program-message-queue parameter.

You can use any positive number that does not exceed the actual number of programs in the program stack, excluding the external message queue.

Message type
INPUT; CHAR(10)
The type of the message being received. The message-type and message-key parameters work together. Depending on the message type, the key can be required, optional, or disallowed. For a list of valid message types and information about how they work with the message-key parameter, see "Message Types and Message Keys" on page 18-13.

Message key
INPUT; CHAR(4)
The key to the message being received. The key is assigned by the command or API that sends the message.

What you can use for the message-key parameter depends on what you use for the message type. For details, see "Message Types and Message Keys" on page 18-13.

If you are not receiving messages by key, use blanks for this parameter.

If you specify a key and the message queue does not contain a message with that key, an error is returned.

If you specify a key and the message queue does contain a message with that key, the API might or might not return a message; it never returns an error in this case. Whether or not the API returns a message depends on the value of the message-type parameter. For example, if you specify the message type *PRV (previous) and the queue does not contain a message before the message with the key, the API does not return a message. Because the key you specified is valid, the API does not return an error either.

You can receive the reply to a message through the key to the sender's copy of the message. If the reply is not available, no message is returned, and the API does not return an error.

When the message type is the special value *NEXT, you can use the special value *TOP for the message key. *TOP returns the message at the top of the queue.

When the message type is the special value *NEXT or *PRV (previous), you can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation.

Wait time
INPUT; BINARY(4)
The length of time to wait for the message to arrive in the queue so it can be received. If the message is not in the queue, the API waits this long and then tries to receive the message again. Valid values are:

0 Do not wait for the message. If the message is not in the queue and you specified a message key, an error is returned.

−1 Wait until the message arrives in the queue and is received, no matter how long it takes. The system has no limit for the wait time.

n (any positive number)
Wait **n** seconds for the message to arrive in the queue.

If you specify a value of zero or above and the message does not arrive in the queue in the specified time, most fields in the RCVM0100 or RCVM0200 output returned by the API are unchanged. The bytes-returned output field has a value of 8, and the bytes-available output field has a value of 0. The remaining output fields are unchanged; they contain whatever was already stored in the space.

Message action
INPUT; CHAR(10)
The action to take after the message is received. Valid values are:

***OLD** Keep the message in the message queue and mark it as an old message. You can receive the message again only by using the message key or by specifying the message type *NEXT, *PRV (previous), *FIRST, or *LAST.

***REMOVE** Remove the message from the message queue. This instance of the message is no longer available for you to work with. The message key is no longer valid; therefore, you cannot receive the message again.

***SAME** Keep the message in the message queue without changing its new or old designation. *SAME lets you receive the message again later without using the message key.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Message Types and Message Keys

For general information about message types and message keys, see "Message Types and Message Keys" on page 18-7.

Messages of type *EXCP are received in last-in first-out (LIFO) order. Messages of all other types are received in first-in first-out (FIFO) order.

The message types you can specify in the QMHRCVPM API are:

Message Type	Message Key	Description
*ANY	Optional	Receives a message of any type except a sender's copy or request.
*COMP	Optional	Receives a completion message.
*COPY	Required	Receives a copy of a previously sent inquiry message. The program message queue parameter must specify the reply message queue specified when the inquiry was sent.
*DIAG	Optional	Receives a diagnostic message.
*ESCAPE	Optional	Receives an escape message.
*EXCP	Optional	Receives an escape or notify message. Both types of messages are received in last-in first-out (LIFO) order.
*FIRST	Disallowed	Receives the first new or old message in the queue, unless it is a request message. Request messages are skipped.
*INFO	Optional	Receives an informational message.
*LAST	Disallowed	Receives the last new or old message in the queue.
*NEXT	Required	Receives the next new or old message after the message with the specified key, unless it is a request message. Request messages are skipped. You can use the special value *TOP for the message key. *TOP designates the message at the top of the message queue. You can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation.
*NOTIFY	Optional	Receives a notify message.
*PRV	Required	Receives the new or old message before the message with the specified key. You can use hexadecimal zeros (hex 00000000) for the message key for the first receive operation.
*RPY	Optional	Receives the reply to an inquiry or notify message. For the message key, you can use the key to the sender's copy of the inquiry or notify message.
*RQS	Optional	Receives the next request in the program message queue for the program. Two actions are possible if no request exists: 1. If the job is interactive and the program queue is the external message queue, the Command Entry display is called to allow the work station user to type requests. 2. In all other cases, an error code or exception is returned.

Error Messages

CPF24AF E Message key not allowed with message type specified.
CPF24A3 E Value for program stack counter not valid.
CPF24A7 E Value for the length of message information not valid.
CPF24A8 E Value for wait time not valid.
CPF24A9 E Value for message action not valid.
CPF24B1 E Message key required for message type specified.
CPF24B2 E Message key of *TOP requires message type of *NEXT.
CPF24B3 E Message type &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF2401 E Not authorized to library &1.
CPF2407 E Message file &1 in &2 not found.
CPF2410 E Message key not found in message queue &1.
CPF2411 E Not authorized to message file &1 in &2.
CPF2415 E End of requests.
CPF2477 E Message queue &1 currently in use.
CPF2479 E Program message queue &1 not found.
CPF2531 E Message file &1 in &2 damaged for &3.
CPF2548 E Damage to message file &1 in &2.
CPF2551 E Message key and message type combination not valid.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9830 E Cannot assign library &1.

Remove Nonprogram Messages (QMHRMVM) API

Parameters

Required Parameter Group:

1	Qualified message queue name	Input	Char(20)
2	Message key	Input	Char(4)
3	Messages to remove	Input	Char(10)
4	Error code	I/O	Char(*)

The Remove Nonprogram Messages (QMHRMVM) API removes messages from nonprogram message queues. The removed instances of the messages are no longer available for you to work with. However, other instances of the messages might still exist in other message queues, and the definitions of predefined messages are still in the message files.

You can use this API to remove a single message from a queue, remove a group of messages, or to clear a message queue of all messages except unanswered inquiries.

To remove messages from program message queues, see "Remove Program Messages (QMHRMVPM) API."

Authorities and Locks

Message Queue Authority *CHANGE
Message Queue Library Authority *USE

Required Parameter Group

Qualified message queue name
INPUT; CHAR(20)
The name of the message queue from which to remove messages, and the library in which it resides. The first 10 characters specify the message queue, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Message key
INPUT; CHAR(4)
If the messages-to-remove parameter specifies *BYKEY, the key to the single message being removed. The key is assigned by the command or API that sends the message.

If the messages-to-remove parameter does not specify *BYKEY, use blanks for this parameter.

Messages to remove
INPUT; CHAR(10)
The message or group of messages being removed. Valid values are:

***ALL**
All messages in the message queue.
***BYKEY**
The single message specified in the message-key parameter.
***KEEPUNANS**
All messages in the message queue except unanswered inquiry messages and unanswered senders' copies.
***NEW**
All new messages in the message queue. New messages are those that have not been received.
***OLD**
All old messages in the message queue. Old messages are those that have already been received.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24AE	E	Message key and messages to remove are mutually dependent.
CPF24A6	E	Value for messages to remove not valid.
CPF24B4	E	Severe error while addressing parameter list.
CPF2401	E	Not authorized to library &1.
CPF2403	E	Message queue &1 in &2 not found.
CPF2408	E	Not authorized to message queue &1.
CPF2410	E	Message key not found in message queue &1.
CPF2450	E	Work station message queue &1 not allocated to job.
CPF2451	E	Message queue &1 is allocated to another job.
CPF2477	E	Message queue &1 currently in use.
CPF3CF1	E	Error code parameter not valid.
CPF8127	E	&8 damage on message queue &4 in &9. VLIC log-&7.
CPF8176	E	Message queue for device description &4 damaged.
CPF9830	E	Cannot assign library &1.

Remove Program Messages (QMHRMVPM) API

Parameters

Required Parameter Group:

1	Program message queue	Input	Char(10)
2	Program stack counter	Input	Binary(4)
3	Message key	Input	Char(4)
4	Messages to remove	Input	Char(10)
5	Error code	I/O	Char(*)

The Remove Program Messages (QMHRMVPM) API removes messages from program message queues and the external message queue. The removed instances of the messages are no longer available for you to work with. However, other instances of the messages might still exist in other message queues, and the definitions of predefined messages are still in the message files.

To remove messages from nonprogram message queues, see "Remove Nonprogram Messages (QMHRMVM) API" on page 18-13.

You can use the QMHRMVPM API to streamline program message handling. Assume that a program receives several diagnostic messages and an escape message from a called program. The program handles the escape message without receiving the diagnostic messages. The program could then use the QMHRMVPM API to remove all the new messages it has not received, including the unneeded diagnostic messages. This can prevent confusion if the program gets another escape message and needs to receive the diagnostic messages associated with the new escape to analyze the error.

Authorities and Locks

Message File Authority *USE
Message File Library Authority *USE

Required Parameter Group

Program message queue
INPUT; CHAR(10)
The name of the program message queue from which to remove messages, or the name of the program to start counting from when using the program-stack-counter parameter. The program you specify must be in the program stack. You can specify a program by name or use one of these special values:

*	The message queue of the current program (that is, the program removing the messages).
***ALLINACT**	All message queues for inactive programs. The messages-to-remove parameter must specify *ALL. The program-stack-counter parameter is ignored.
***EXT**	The external message queue. The program-stack-counter parameter is ignored.

If you specify a message key and the messages-to-remove parameter specifies *BYKEY, this parameter is ignored.

Program stack counter
INPUT; BINARY(4)
A number identifying the location in the program stack of the program message queue from which to remove messages. The number is relative to the program specified in the program-message-queue parameter. It indicates how many calls up the program stack the targeted program message queue is from the one specified in the program-message-queue parameter. Valid values are:

0 Remove the messages from the queue of the program specified in the program-message-queue parameter.

1 Remove the messages from the queue of the program that calls the program specified in the program-message-queue parameter.

n (any positive number)
Remove the messages from the queue of the nth program up the stack from the program specified in the program-message-queue parameter.

You can use any positive number that does not exceed the actual number of programs in the program stack. Do not include the external message queue in your count.

This parameter is ignored in these cases:

- When the program-message-queue parameter specifies all queues for inactive programs (*ALLINACT) or the external message queue (*EXT).
- When you specify a message key and the messages-to-remove parameter specifies *BYKEY.

Message key
INPUT; CHAR(4)
If the messages-to-remove parameter specifies *BYKEY, the key to the single message being removed. (The key is assigned by the command or API that sends the message.) The program-message-queue and program-stack-counter parameters are ignored.

If the messages-to-remove parameter does not specify *BYKEY, you must use blanks for this parameter.

Messages to remove
INPUT; CHAR(10)
The message or group of messages being removed. Valid values are:

***ALL**
All messages in the program message queue.

***BYKEY**
The single message specified in the message-key parameter.

***KEEPRQS**
All messages in the program message queue except requests.

***NEW**
All new messages in the program message queue. New messages are those that have not been received.

***OLD**
All old messages in the program message queue. Old messages are those that have already been received.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24AD E Messages to remove must be *ALL if program message queue is *ALLINACT.

CPF24AE E Message key and messages to remove are mutually dependent.
CPF24A3 E Value for program stack counter not valid.
CPF24A6 E Value for messages to remove not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF2410 E Message key not found in message queue &1.
CPF2467 E &3 message queue &1 in library &2 logically damaged.
CPF2477 E Message queue &1 currently in use.
CPF2479 E Program message queue &1 not found.
CPF3CF1 E Error code parameter not valid.
CPF8127 E &8 damage on message queue &4 in &9. VLIC log-&7.

Resend Escape Message (QMHRSNEM) API

Parameters

Required Parameter Group:

1	Message key	Input	Char(4)
2	Error code	I/O	Char(*)

The Resend Escape Message (QMHRSNEM) API resends an escape message from the current program's message queue to the previous program's message queue. Moving a message in this way does not change the sender information stored with the message.

You can use this API along with the Move Program Messages (QMHMOVPM) API to streamline program message handling. If a program is sent diagnostic messages and an escape message but cannot handle the error itself, you can use the QMHMOVPM API to move the diagnostic messages and the QMHRSNEM API to forward the escape message to the previous program in the program stack. The program does not need to send an escape message of its own. For details about the QMHMOVPM API, see "Move Program Messages (QMHMOVPM) API" on page 18-4.

Required Parameter Group

Message key
INPUT; CHAR(4)
The key to the escape message being resent. The key is assigned by the command or API that first sends the message.

To resend the last new escape message, use blanks for this parameter.

A message is new until it is received. It then becomes an old message. You can resend an old message only if you specify the message key.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF2410 E Message key not found in message queue &1.
CPF2460 E Message queue &1 could not be extended.
CPF2477 E Message queue &1 currently in use.
CPF3CF1 E Error code parameter not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.

Retrieve Message (QMHRTVM) API

Parameters

Required Parameter Group:

1	Message information	Output	Char(*)
2	Length of message information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Message identifier	Input	Char(7)
5	Qualified message file name	Input	Char(20)
6	Message data	Input	Char(*)
7	Length of message data	Input	Binary(4)
8	Replace substitution values	Input	Char(10)
9	Return format control characters	Input	Char(10)
10	Error code	I/O	Char(*)

The Retrieve Message (QMHRTVM) API retrieves the message definition of a predefined message. The **message definition** is created with the Add Message Description (ADDMSGD) command. It consists of the text of the message and other information, such as the message help and the default reply for the message. You can use the QMHRTVM API to copy the text of predefined messages into a program.

Retrieving a message is not the same as receiving a message. Retrieving a message with this API returns the message and associated information stored in the message file. In contrast, receiving a message with the Receive Nonprogram Message (QMHRCVM) or Receive Program Message (QMHRCVPM) API focuses on the message as it is sent to a particular user or program at a particular time; the information returned includes details about who sent the message, when it was sent, why it was sent, and so on.

Authorities and Locks

Message File Authority *USE
Message File Library Authority *USE

Required Parameter Group

Message information
OUTPUT; CHAR(*)
The variable that receives information returned, in the format specified in the format-name parameter, of the length specified in the length-of-message-information parameter.

Length of message information
INPUT; BINARY(4)
The size of the area to contain the message information, in bytes. The minimum size is 8.

This parameter must specify the size of the variable you use for the message-information parameter. If this parameter specifies a longer size, other parts of storage could be overwritten when the API returns the information.

The API returns as much information as it can fit in this length. If the available message information is longer, it is truncated. If the available message information is shorter, the unused output area is unchanged; whatever is already stored in that space remains there.

To determine how much information the API actually returns in response to this call, see the bytes-returned field in the RTVM0100 or RTVM0200 output. To determine how much information the API could return if space were available, see the bytes-available field.

Format name
INPUT; CHAR(8)
The format to use for the message information. Specify one of these format names:

RTVM0100 Brief message information. For details, see "RTVM0100 Format."
RTVM0200 All message information. For details, see "RTVM0200 Format" on page 18-18.

Message identifier
INPUT; CHAR(7)
The identifying code for the predefined message being retrieved.

Qualified message file name
INPUT; CHAR(20)
The name of the message file from which to retrieve the message information, and the library in which it resides. The first 10 characters specify the file name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Message data
INPUT; CHAR(*)
The values to insert in the substitution variables in the predefined message and message help.

If you use blanks for this parameter, blanks are inserted for the message data.

If this parameter contains pointer data, each pointer must start on a 16-byte boundary to keep the data accurate.

Length of message data
INPUT; BINARY(4)
The length of the data, in bytes. Valid values are 0 through 32 767.

Replace substitution variables
INPUT; CHAR(10)
Whether to replace the substitution variables with the values given in the message-data parameter. Specify one of these values:

***NO** Do not use the message data. Instead, return the substitution variable numbers (that is, &1, &2, and so on) in the message text.
***YES** Use the message data in the message text. If you specify blanks in the message-data parameter, blanks are used for the substitution variables.

Return format control characters
INPUT; CHAR(10)
Whether or not the format control characters are returned in the message help output field. Specify one of these values:

***NO** Do not return the format control characters in the text.
***YES** Return the format control characters in the text.

Format control characters are codes like &B and &P inserted in the text of the message help. They help determine the format of the message help when it is displayed. For more information, see the Add Message Description (ADDMSGD) command in the *CL Reference*.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

RTVM0100 Format

The following table lists the fields in the RTVM0100 format. For more information about each field, see "Field Descriptions" on page 18-18.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Length of message returned
12	C	BINARY(4)	Length of message available
16	10	BINARY(4)	Length of message help returned
20	14	BINARY(4)	Length of message help available
24	18	CHAR(*)	Message

Offset		Type	Field
Dec	Hex		
The offset to this field equals the offset to the last fixed-length field plus the length of the previous variable-length fields.		CHAR(*)	Message help

RTVM0200 Format

The following table lists the fields in the RTVM0200 format. For more information about each field, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Message severity
12	C	BINARY(4)	Alert index
16	10	CHAR(9)	Alert option
25	19	CHAR(1)	Log indicator
26	1A	CHAR(2)	Reserved
28	1C	BINARY(4)	Length of default reply returned
32	20	BINARY(4)	Length of default reply available
36	24	BINARY(4)	Length of message returned
40	28	BINARY(4)	Length of message available
44	2C	BINARY(4)	Length of message help returned
48	30	BINARY(4)	Length of message help available
52	34	CHAR(*)	Default reply
The offsets to these fields equal the offset to the last fixed-length field plus the length of the previous variable-length fields.		CHAR(*)	Message
		CHAR(*)	Message help

Field Descriptions

This section describes the fields returned in further detail. The fields are listed in alphabetical order.

Alert index. The format number for the message data field. This number is also called the resource name variable. For more information, see the *Alerts and DSNX Guide*.

Alert option. Whether and when an SNA alert is created and sent for the message. Valid values are:

**DEFER*	An alert is sent after local problem analysis.
**IMMED*	An alert is sent immediately, and the message is sent to the QHST message log at the same time.
**NO*	No alert is sent.
**UNATTEND*	An alert is sent immediately when the system is running in unattended mode (that is, when the value of the alert status network attribute, ALRSTS, is *UNATTEND).

For more information, see the *Alerts and DSNX Guide*.

Bytes available. The length of all available information about the format. Bytes available can be greater than the length specified in the API's length-of-message-information parameter. If it is greater, the information returned is truncated.

Bytes returned. The length of all information returned in the format. The value of the bytes-returned field is always less than or equal to the value of the length-of-message-information parameter, and less than or equal to the bytes available. If the bytes-returned value is less than the length-of-message-information value, the unused space is unchanged.

Default reply. The default reply for the message identifier retrieved.

Length (general information about the following length fields). For a more detailed description of the length fields, see "Field Descriptions" on page 18-9.

Length of default reply available. The length of the available default reply, in bytes.

Length of default reply returned. The length of the returned default reply, in bytes.

Length of message available. The length of the available message text, in bytes.

Length of message help available. The length of the available help information for the message, in bytes.

Length of message help returned. The length of the returned help information for the message, in bytes.

Length of message returned. The length of the returned message text, in bytes.

Log indicator. The log problem indicator for the message retrieved. Possible values are:

N Problems are not logged.
Y Problems are logged.

Message. The text of the message retrieved.

Message help. The message help for the message retrieved.

Message severity. The severity of the message retrieved.

Reserved. An ignored field.

Error Messages

CPF24AA E Value for replace substitution variables not valid.
CPF24AB E Value for return format control characters not valid.
CPF24A7 E Value for the length of message information not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B6 E Length of text or data not valid.
CPF2401 E Not authorized to library &1.
CPF2407 E Message file &1 in &2 not found.
CPF2411 E Not authorized to message file &1 in &2.
CPF2419 E Message identifier &1 not found in message file &2 in &3.
CPF2465 E Replacement text of message &1 in &2 in &3 not valid for format specified.
CPF2499 E Message identifier &1 not allowed.
CPF2531 E Message file &1 in &2 damaged for &3.
CPF2548 E Damage to message file &1 in &2.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9830 E Cannot assign library &1.

Retrieve Request Message (QMHRTVRQ) API

Parameters

Required Parameter Group:

1	Message information	OUTPUT	CHAR(*)
2	Length of message information	INPUT	BINARY(4)
3	Format name	INPUT	CHAR(8)
4	Message type	INPUT	CHAR(10)
5	Message key	INPUT	CHAR(4)
6	Error code	I/O	CHAR(*)

The Retrieve Request Message (QMHRTVRQ) API retrieves request messages from the current job's job message queue. Only request messages (commands) that have been received are retrieved; new request messages and all other types of messages are bypassed. One use for this API is in programs that support a retrieve key on a command line to retrieve commands previously run.

Required Parameter Group

Message information
OUTPUT; CHAR(*)
The variable that is to receive information about the request message. The minimum size for this area is 8 bytes. If the size of this area is smaller than the available message information, the API returns only the data that the area can hold. If no request message could be retrieved from the job message queue (for example, there are no request messages prior to or after the one identified by the message key passed in), no error will be returned. Instead, bytes available will be set to 0 to indicate no message was found.

Length of message information
INPUT; BINARY(4)
The size, in bytes, of the area to contain the message information. The minimum size is 8. If this value is larger than the actual size of the storage allocated for the message information parameter, unpredictable results may occur.

Format name
INPUT; CHAR(8)
The format of the message information to be returned. Valid formats are:

RTVQ0100 Basic request message information.
RTVQ0200 All request message information.

Message type
INPUT; CHAR(10)
The message to be retrieved. The valid values are:

***FIRST** Retrieve the first request message in the current job.
***LAST** Retrieve the last request message in the current job.
***NEXT** Retrieve the request message after the message indicated by the message key parameter. Message key is required when *NEXT is used.
***PRV** Retrieve the request message before the message indicated by the message key parameter. Message key is required when *PRV is used.

Message key
INPUT; CHAR(4)
A value must be specified for this parameter when the message type parameter is *NEXT or *PRV and is used to retrieve the request message after or before the message with this key. This message key must refer to a message on the job message queue. This value must be blank when the message type parameter is *FIRST or *LAST.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

RTVQ0100 Format

The following table describes the information that is returned in the message information parameter for the RTVQ0100 format. For detailed descriptions of the fields, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(4)	Message key
12	C	CHAR(24)	Reserved
36	24	BIN(4)	Length of request message text
40	28	CHAR(*)	Request message text

RTVQ0200 Format

The following table describes the information that is returned in the message information parameter for the RTVQ0200 format. For detailed descriptions of the fields, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(4)	Message key
12	C	CHAR(10)	Program name
22	16	CHAR(278)	Reserved
300	12C	BIN(4)	Length of request message text
304	130	CHAR(*)	Request message text

Field Descriptions

Bytes available. The length of all data available to return. All available data is returned if enough space is provided.

Bytes returned. The length of all data actually returned. If the data is truncated because the length of message information parameter did not specify an area large enough to hold all of the data available, this value will be less than bytes available. When no request messages could be retrieved from the job message queue, the value of the bytes returned field is set to 8, the value of the bytes available field is set to 0, and the remaining fields are unchanged.

Length of request message text. The length of all available request message text, in bytes. If the message information parameter is not large enough to hold all of the request message text available, the amount of request message text actually returned will be less than this value.

Message key. The key to the request message retrieved. This value can be used on subsequent calls to this API using *NEXT or *PRV in the message type parameter to retrieve the next or previous request message on the job message queue. It can also be used on the Receive Program Message (QMHRCVPM) API to get more information about the request message.

Program name. The name of the program to where the request message was sent.

Request message text. The text of the request message. If you are retrieving CL commands, the current maximum length of a CL command is 6000 bytes.

Reserved. This field is reserved for future expansion.

Error Messages

CPF24AF E Message key not allowed with message type specified.
CPF24A7 E Value for the length of message information not valid.
CPF24B3 E Message type &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF2410 E Message key not found in message queue &1.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.

Send Break Message (QMHSNDBM) API

Parameters

Required Parameter Group:

1	Message text	Input	Char(*)
2	Length of message text	Input	Binary(4)
3	Message type	Input	Char(10)
4	List of qualified messages	Input	Array of Char(20)
5	Number of message queues	Input	Binary(4)
6	Qualified name of the reply message queue	Input	Char(20)
7	Error code	I/O	Char(*)

The Send Break Message (QMHSNDBM) API sends an immediate message to a work station message queue and forces the message queue to be displayed, interrupting whatever is on the user's display. A message delivered in this way is called a **break message**. Use break messages to ensure that users see important information, such as a warning to sign off before you shut the system down for maintenance.

To keep break messages from interrupting a job, change the value of the BRKMSG parameter of the Change Job (CHGJOB) command.

If your application attempts to diagnose and recover from errors, it might need to take additional action when using the QMHSNDBM API. The QMHSNDBM API sends a message to a list of message queues. When the API encounters an error in sending your message to a message queue in the list, it sends a diagnostic message describing that error to your program message queue, and then proceeds to the next message queue in the list. After trying to send the message to all specified message queues and if the API detected any errors, it returns a general escape message CPF2469 as an exception or in the error code.

To diagnose and recover from these errors, your program should call the QMHRCVPM API to receive the diagnostic messages sent.

Required Parameter Group

Message text
INPUT; CHAR(*)
The complete text of the immediate message being sent. You cannot include pointer data in this parameter.

Length of message text
INPUT; BINARY(4)
The length of the message text, in bytes. Valid values are 1 through 512.

Message type
INPUT; CHAR(10)
The type of the message. You must specify one of these values:

***INFO** Informational. Conveys information *without* asking for a reply.
***INQ** Inquiry. Conveys information *and* asks for a reply.

List of qualified message queue names
INPUT; ARRAY of CHAR(20)
A list of 1 through 50 work station message queues to which the message is being sent, and the libraries in which they reside.

When you send an informational message, you can specify 1 through 50 specific message queues or use this special value for the message queue name:

***ALLWS** All work station message queues. Use blanks for the library name, do not specify any other message queues, and specify 1 for the number-of-message-queues parameter.

When you send an inquiry message, specify only one message queue.

When you send an informational or inquiry message to a specific message queue, use the first 10 characters for the message queue name and the second 10 characters for the library name. You can use this special value for the library name:

***LIBL** The library list

If you send a break message to a specific work station message queue and that work station is not logged on, no error is returned. However, diagnostic message CPF2429, `Message to work station did not break`, appears in the job log.

Number of message queues
INPUT; BINARY(4)
The number of message queues specified in the list-of-qualified-message-queue-names parameter.

For informational (*INFO) messages, valid values are 1 through 50.

For inquiry (*INQ) messages or when using the special value *ALLWS for the message queue, you must specify 1.

Qualified name of the reply message queue
INPUT; CHAR(20)
For an inquiry message, the name of the message queue to receive the reply message, and the library in which it resides. The first 10 characters specify the message queue, and the second 10 characters specify the library.

You can use this special value for the message queue name:

***PGMQ** The current program's message queue. Use blanks for the library name.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

For an informational message, use blanks for this parameter. If you specify a message queue, the API ignores it and does not return an error.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24A2 E Value for number of message queues not valid.
CPF24B3 E Message type &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B6 E Length of text or data not valid.
CPF2421 E Message not sent. &1 in &2 not work station message queue.
CPF2428 E Only one message queue allowed for *INQ and *NOTIFY type messages.
CPF2401 D Not authorized to library &1.
CPF2403 D Message queue &1 in &3 not found.
CPF2408 D Not authorized to message queue &1.

CPF2421 D Message not sent. &1 in &2 not work station message queue.
CPF2429 D Message to work station &1 did not break.
CPF2460 D Message queue could not be extended.
CPF2467 D &3 message queue &1 in library &2 logically damaged.
CPF2477 D Message queue &1 currently in use.
CPF2481 D Work station message queue not available.
CPF8100 D All CPF81xx damage notification messages.
CPF2469 E Error occurred when sending message&1.
CPF3CF1 E Error code parameter not valid.
CPF9830 E Cannot assign library &1.

Send Nonprogram Message (QMHSNDM) API

Parameters

Required Parameter Group:

1	Message identifier	Input	Char(7)
2	Qualified message file name	Input	Char(20)
3	Message data or immediate text	Input	Char(*)
4	Length of message data or immediate text	Input	Binary(4)
5	Message type	Input	Char(10)
6	List of qualified message queue names	Input	Array of Char(20)
7	Number of message queues	Input	Binary(4)
8	Qualified name of the reply message queue	Input	Char(20)
9	Message key	Output	Char(4)
10	Error code	I/O	Char(*)

The Send Nonprogram Message (QMHSNDM) API sends a message to a nonprogram message queue so your program can communicate with another job or user.

To send a message to a program message queue or the external message queue, see "Send Program Message (QMHSNDPM) API" on page 18-24.

Before coding your call to the QMHSNDM API, see "Dependencies among Parameters" on page 18-23.

If your application attempts to diagnose and recover from errors, it might need to take additional action before calling the QMHSNDM API. The QMHSNDM API sends a message to a list of message queues. When the API encounters an error in sending your message to a message queue in the list, it sends a diagnostic message describing that error to your program message queue, and then proceeds to the next message queue in the list. After trying to send the message to all specified message queues and if the API detected any errors, it returns the general escape message CPF2469 as an exception or in the error code.

To diagnose and recover from these errors, your program should call the QMHRCVPM API to receive the diagnostic messages sent.

For an example program using the QMHSNDM API, see "Diagnostic Reporting" on page A-19.

Authorities and Locks

Message File Authority	*USE
Message File Library Authority	*USE
Message Queue Authority	*CHANGE

Required Parameter Group

Message identifier
INPUT; CHAR(7)
The identifying code for the predefined message being sent, or blanks for an immediate message.

If you specify a message identifier, you must specify a qualified message file name. If you do not specify a message identifier, the qualified-message-file-name parameter is ignored.

Qualified message file name
INPUT; CHAR(20)
For a predefined message, the name of the message file and the library in which it resides. The first 10 characters specify the file name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

For an immediate message, use blanks for this parameter. If you specify a name, the API ignores it and does not return an error.

Message data or immediate text
INPUT; CHAR(*)
If a message identifier is specified, the data to insert in the predefined message's substitution variables. If no message identifier is specified, the complete text of an immediate message.

If this parameter contains pointer data, each pointer must start on a 16-byte boundary to keep the data accurate.

Length of message data or immediate text
INPUT; BINARY(4)
The length of the message data or immediate text, in bytes. Valid values for each are:

Message data 0 – 32 767
Immediate text 1 – 512

Message type
INPUT; CHAR(10)
The type of the message. You must specify one of these values:

***COMP** Completion
***DIAG** Diagnostic
***INFO** Informational

***INQ** Inquiry. When you send an inquiry message, a copy of the message is placed on the reply message queue, and the message key returned by this API is the key to that copy. To receive the reply, use the "Receive Nonprogram Message (QMHRCVM) API" on page 18-5, specifying that key and a message type of reply. To receive the copy of the inquiry, specify the same key and a message type of copy.

To send reply messages, see "Send Reply Message (QMHSNDRM) API" on page 18-28.

For descriptions of the message types, see "Message Types" on page 18-2. For details about coding the other parameters when sending a particular type of message, see "Dependencies among Parameters."

List of qualified message queue names
INPUT; ARRAY of CHAR(20)
A list of 1 through 50 message queues to which the message is being sent, and the libraries in which they reside. When sending an inquiry message or using the special value *ALLACT, you can list only one queue. In all other cases, you can list up to 50 message queues.

You can specify user profile message queues or other nonprogram message queues, or you can use several special values.

To specify the default message queue associated with a user profile, use the first 10 characters for the user profile name. In the second 10 characters, use the special value *USER.

To specify other nonprogram message queues, use the first 10 characters for the message queue name and the second 10 characters for the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

To specify other types of message queues, use one of the following special values for the first 10 characters, and leave the second 10 characters blank:

***ALLACT**
The default message queues of all active users. When you use this value, it must be the only item in the list. You cannot use this value with inquiry messages.
***REQUESTER**
In an interactive job, the current user's message queue. In a batch job, the system operator's message queue, QSYSOPR.
***SYSOPR**
The system operator's message queue, QSYSOPR.

Number of message queues
INPUT; BINARY(4)
The number of message queues specified in the list-of-qualified-message-queue-names parameter. Valid values are 1 through 50. When sending an inquiry message or using the special value *ALLACT for the message queue, you must specify 1.

Qualified name of the reply message queue
INPUT; CHAR(20)
For an inquiry message only, the name of the message queue to receive the reply message, and the library in which it resides.

The first 10 characters specify the message queue, and the second 10 characters specify the library.

You can use these special values for the message queue name:

***PGMQ** The current program's message queue. Use blanks for the library name.
***WRKSTN** The work station message queue. Use blanks for the library name. You cannot use this value when running in batch mode.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

For all message types except inquiry, use blanks for this parameter. If you specify a message queue, the API ignores it and does not return an error.

Message key
OUTPUT; CHAR(4)
For an inquiry message only, the key to the sender's copy of the message in the reply message queue. This API assigns the key when it sends the message.

For all other message types, no key is returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Dependencies among Parameters

You can use many different combinations of parameters when calling the QMHSNDM API. The values you specify for the message-identifier and message-type parameters determine which other input parameters you must specify and which you must code as blanks. They also determine whether the system returns the message key in the message-key output parameter.

The following table summarizes the use of parameters for the QMHSNDM API. The table uses these terms to describe the parameter values:

Term	Meaning
Blank	You must use blanks for this parameter.
Data	You must specify data for this parameter. The data is used as the values for a predefined message's substitution variables.
Ignored	The API ignores this parameter. You should use blanks for the value, but if you use another value, no error is returned.
Message ID	You must specify the message identifier of the predefined message being sent.
Not used	The API does not return data in this output parameter. The space remains unchanged.
Required	You must specify a valid value for this parameter.
Returned	The API returns data in this output parameter.
Text	You must specify text for this parameter. The text is used as the complete text of an immediate message.

Parameter	Message Type			
	*COMP, *DIAG, *INFO		*INQ	
	Predefined	Immediate	Predefined	Immediate
Message identifier	Message ID	Blank	Message ID	Blank
Qualified message file name	Required	Ignored	Required	Ignored
Message data or immediate text	Data	Text	Data	Text
Length of message data or immediate text	0-32 767	1-512	0-32 767	1-512
List of qualified message queue names	Required	Required	Required	Required
Number of message queues	1-50	1-50	1	1
Reply message queue	Ignored	Ignored	Required	Required
Message key	Not used	Not used	Returned	Returned
Error code	Required	Required	Required	Required

Error Messages

CPF2204 E User profile &1 not found.
CPF24AC E Either message identifier or message text must be specified.
CPF24A2 E Value for number of message queues not valid.
CPF24B3 E Message type &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B6 E Length of text or data not valid.
CPF2428 E Only one message queue allowed for *INQ and *NOTIFY type messages.
CPF2433 E Function not allowed for system log message queue &1.
CPF2469 E Error occurred when sending message &1.
CPF2401 D Not authorized to library &1.
CPF2403 D Message queue &1 in &3 not found.
CPF2407 D Message file &1 in &2 not found.
CPF2408 D Not authorized to message queue &1.
CPF2411 D Not authorized to message file &1 in &2.
CPF2435 D System reply list not found.
CPF2460 D Message queue &1 could not be extended.
CPF2467 D &3 message queue &1 in library &2 logically damaged.
CPF2477 D Message queue &1 currently in use.
CPF2531 D Message file in &2 damaged for &3.
CPF2548 D Damage to message file &1 in &2.
CPF2557 D System reply list damaged.
CPF2558 D System reply list currently in use.
CPF8100 D All CPF81xx damage notification messages.
CPF9830 D Cannot assign library &1.
CPF2481 E Work station message queue not available.
CPF2488 E Reply message queue *WRKSTN not valid for batch job.
CPF2499 E Message identifier &1 not allowed.
CPF3CF1 E Error code parameter not valid.
CPF9830 E Cannot assign library &1.

Parameters

Required Parameters Group:

1	Message identifier	Input	Char(7)
2	Qualified message file name	Input	Char(20)
3	Message data or immediate text	Input	Char(*)
4	Length of message data or immediate text	Input	Binary(4)
5	Message type	Input	Char(10)
6	Program message queue	Input	Char(10)
7	Program stack counter	Input	Binary(4)
8	Message key--Output	Output	Char(4)
9	Error code	I/O	Char(*)

The Send Program Message (QMHSNDPM) API sends a message to a program message queue or the external message queue. (The external message queue is the part of the job message queue that handles messages between an interactive job and the work station user. It is not associated with a specific program.) This API allows the current program to send a message to its caller, a previous caller, or itself.

To send a message to a nonprogram message queue, see "Send Nonprogram Message (QMHSNDM) API" on page 18-22.

Before coding your call to the QMHSNDPM API, see "Dependencies among Parameters" on page 18-26.

Authorities and Locks

Message File Authority *USE
Message File Library Authority *USE

Required Parameter Group

Message identifier
INPUT; CHAR(7)
The identifying code for the predefined message being sent, or blanks for an immediate message.

When sending an escape, notify, or status message, you must specify a message identifier. When sending a request message, you must use blanks. When sending other types of messages, you can use either a message identifier or blanks.

If you specify a message identifier, you must specify a qualified message file name. If you do not specify a message identifier, the API ignores the qualified-message-file-name parameter.

Qualified message file name
INPUT; CHAR(20)
For a predefined message, the name of the message file and the library in which it resides. The first 10 characters specify the file name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

For an immediate message, use blanks for this parameter. If you specify a name, the API ignores it and does not return an error.

Message data or immediate text
INPUT; CHAR(*)
If a message identifier is specified, this parameter specifies the data to insert in the predefined message's substitution variables. If blanks are used for the message identifier, this parameter specifies the complete text of an immediate message.

If this parameter contains pointer data, each pointer must start on a 16-byte boundary to keep the data accurate.

Length of message data or immediate text
INPUT; BINARY(4)
The length of the message data or immediate text, in bytes. Valid values for each are:

Message data 0 – 32 767
Immediate text 1 – 512

Message type
INPUT; CHAR(10)
The type of the message. You must specify one of these values:

***COMP** Completion
***DIAG** Diagnostic
***ESCAPE** Escape
***INFO** Informational
***INQ** Inquiry. You can send inquiry messages only to the external message queue.
***NOTIFY** Notify
***RQS** Request
***STATUS** Status

To send reply messages, see "Send Reply Message (QMHSNDRM) API" on page 18-28.

For descriptions of the message types, see "Message Types" on page 18-2. For details about coding the other parameters when sending a particular type of message, see "Dependencies among Parameters" on page 18-26. For further information about working with each type, see "Additional Information about Message Types" on page 18-26.

Program message queue
INPUT; CHAR(10)
The name of the program message queue to send the messages to, or the name of the program to start counting from when using a value other than 0 for the program-stack-counter parameter. The program you specify must be in the program stack. You can specify a program by name or use one of these special values:

* The message queue of the current program (that is, the program sending the message).

***EXT** The external message queue. The program-stack-counter parameter is ignored. You cannot send escape messages to this message queue. You can send inquiry messages to this message queue only.

Program stack counter
INPUT; BINARY(4)
A number identifying the location in the program stack of the program message queue to which messages are sent. The number is relative to the program specified in the program-message-queue parameter. It indicates how many calls up the program stack the program message queue to which you send the message is from the one specified in the program-message-queue parameter.

Valid values are:

0 Send the message to the queue of the program specified in the program-message-queue parameter.

1 Send the message to the queue of the program that calls the program specified in the program-message-queue parameter.

n (any positive number)
Send the message to the queue of the nth program up the stack from the program specified in the program-message-queue parameter.

You can use any positive number that does not exceed the actual number of programs in the program stack, excluding the external message queue.

Message key
OUTPUT; CHAR(4)
The key to the message being sent. This API assigns the key when it sends a message of any type except escape or status. For an escape or status message, no key is returned and this parameter is not changed.

For inquiry and notify messages, this is the key to the sender's copy of the message in the sending program's message queue.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Dependencies among Parameters

You can use many different combinations of parameters when calling the QMHSNDPM API. The values you specify for the message-identifier and message-type parameters determine which other input parameters you must specify and which you must code as blanks. They also determine whether the system returns the message key in the message-key output parameter.

The following table summarizes the use of parameters for the QMHSNDPM API. For definitions of the terms used to describe the parameter values, see the list on page 18-24.

Parameter	Message Type						
	*COMP, *DIAG, *INFO		*INQ		*ESCAPE, *STATUS	*NOTIFY	*RQS
	Predefined	Immediate	Predefined	Immediate	Predefined	Predefined	Immediate
Message identifier	Message ID	Blank	Message ID	Blank	Message ID	Message ID	Blank
Qualified message file name	Required	Ignored	Required	Ignored	Required	Required	Ignored
Message data or immediate text	Data	Text	Data	Text	Data	Data	Text
Length of message data or immediate text	0-32 767	1-512	0-32 767	1-512	0-32 767	0-32 767	1-512
Program message queue	Required	Required	*EXT	*EXT	Required	Required	Required
Program stack counter	0-n	0-n	Ignored	Ignored	0-n	0-n	0-n
Message key	Returned	Returned	Returned	Returned	Not used	Returned	Not used
Error code	Required	Required	Required	Required	Required	Required	Required

Additional Information about Message Types

The following sections discuss additional considerations in using the different types of messages that you can specify when calling the QMHSNDPM API.

Completion and Diagnostic Messages: You can send completion and diagnostic messages as immediate messages or with message identifiers to program message queues or to the external message queue.

Escape Messages: You can send escape messages only with a message identifier and only to a program message queue. You cannot send them to the external message queue.

An escape message sent to a previous program's message queue ends the current program. Sending the escape message causes all the programs up to but not including the program receiving the escape message to end. For example, if three programs are active and the third program sends an escape message to the first, then the second and third programs end.

The message key is never returned to the sending program. Escape messages indicate that an error prevents the program from continuing to run. The sending program ends as part of sending the escape message. If the sending program sends the escape message to itself and monitors for the escape message, the monitoring routine can take control, but the message key is still not returned.

Informational Messages: You can send informational messages as immediate messages or with message identifiers to program message queues or to the external message queue.

When you send an informational or inquiry message to the external message queue, the message is displayed on the work station of the current job. The user sees the Display Program Messages display. Sending messages of these types to the external message queue is the only way to make the Display Program Messages display appear.

Inquiry Messages: You can send inquiry messages only to the external message queue. You cannot send them to program message queues because the sending program would have to end before the receiving program could receive the message and send a reply. At that point, the sending program is no longer active, so it cannot receive a reply.

When you send an inquiry message, it is displayed at the work station as described in "Informational Messages." In addition, a copy of the message is placed in the reply message queue, which is always the sending program's message queue. Sending an inquiry message lets your program receive the reply to the message by using the message key returned by this API.

An inquiry message sent to the external message queue requires either the default reply or a reply from an interactive user. Programs cannot reply to inquiry messages sent to the external message queue. In batch mode, the default reply is automatically sent as the reply. When the reply is sent, the inquiry message and the reply are automatically removed from the external message queue so that the message appears only once in the job log, as a sender's copy with a reply.

The QMHSNDPM API returns the message key to the copy of the inquiry message in the reply message queue. You can receive the reply to the inquiry message by using the Receive Program Message (QMHRCVPM) API specifying this key and the message type reply. You can receive the copy of the original inquiry message by specifying this key and the message type of *COPY. You cannot use this key to receive the original message.

Notify Messages: You can send notify messages only with a message identifier. You can send them to a program message queue or to the external message queue.

The reply message queue is always the sending program's message queue (*PGMQ). This API returns the message key to the copy of the message in the reply message queue. To receive the reply, use the Receive Program Message (QMHRCVPM) API specifying this key and a message type of reply (*RPY).

When a notify message is sent to a program message queue, the action taken depends on whether the receiving program monitors for the message. When the receiving program does monitor for the message, the sending program ends and the receiving program takes control. No default reply is sent. In contrast, when the receiving program does not monitor for the message, the sending program resumes control at the next instruction, and the default reply to the message is automatically sent.

When a notify message is sent to the external message queue, the action taken depends on whether the job is interactive or batch. In an interactive job, the user must take action to send any reply, including the default reply. In a batch job, the default reply is automatically sent because programs cannot reply to notify messages sent to the external message queue. After a reply is sent, the sender's copy of the message and the reply are removed from the reply message queue.

Request Messages: You can send a request message only as an immediate message. You can send it to a program message queue or to the external message queue.

Request messages sent to the external message queue are never displayed on the Display Program Messages display. The system program QCMD receives them. Before the Command Entry display appears, the QCMD program attempts to perform the commands contained in the request messages. The QCMD program performs this function only with request messages in the external message queue; it does not work with request messages in other queues.

Status Messages: You must send status messages with a message identifier. You can send them to a program message queue or the external message queue.

When you send status messages to the external message queue, the system displays them at the bottom of the current job's display station to report the status of running programs. You can use this to reassure users that long programs are still running.

CL programs and programs that run under the extended program model (EPM) can monitor for status messages. For more information about EPM programs, see "Error Handling in the Extended Program Model (EPM) Environment" on page 18-3.

Status messages appear only on the display. Because they are never put in message queues and they do not have message keys, you cannot receive them.

Status messages are predefined messages. However, you can send the user what appears to be an immediate status message by using a message type of status and message CPF9898 in message file QCPFMSG in library QSYS. This message consists of a substitution variable; thus, it uses the message data that you supply as the full message text. To use it, specify CPF9898 in the message-identifier parameter and the text you want to send in the message-data-or-immediate-text parameter.

Error Messages

CPF24AC E Either message identifier or message text must be specified.
CPF24A3 E Value for program stack counter not valid.
CPF24B3 E Message type &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B6 E Length of text or data not valid.
CPF2401 E Not authorized to library &1.
CPF2407 E Message file &1 in &2 not found.
CPF2409 E Specified message type not valid with specified program message queue.
CPF2411 E Not authorized to message file &1 in &2.
CPF2435 E System reply list not found.
CPF2467 E &3 message queue &1 in library &2 logically damaged.
CPF2469 E Error occurred when sending message&1.
CPF2479 E Program message queue &1 not found.
CPF2489 E Message identifier required for *NOTIFY, *ESCAPE, or *STATUS message types.
CPF2493 E Message identifier not allowed with message type *RQS.
CPF2499 E Message identifier &1 not allowed.
CPF2531 E Message file &1 in &2 damaged for &3.
CPF2548 E Damage to message file &1 in &2.
CPF2557 E System reply list damaged.
CPF2558 E System reply list currently in use.
CPF3CF1 E Error code parameter not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9830 E Cannot assign library &1.
CPF9845 E Error occurred while opening file &1.
CPF9846 E Error while processing file &1 in library &2
CPF9847 E Error occurred while closing file &1 in library &2.

Send Reply Message (QMHSNDRM) API

Parameters

Required Parameter Group:

1	Message key	Input	Char(4)
2	Qualified message queue name	Input	Char(20)
3	Reply text	Input	Char(*)
4	Length of reply text	Input	Binary(4)
5	Remove message	Input	Char(10)
6	Error code	I/O	Char(*)

The Send Reply Message (QMHSNDRM) API sends a reply message to the sender of an inquiry message.

If your application attempts to diagnose and recover from errors, it might need to take additional action when using the QMHSNDRM API. When the QMHSNDRM API encounters errors in the reply being sent, it sends a diagnostic message describing each error to your program message queue. After the last diagnostic message, the QMHSNDRM API sends a general escape message, CPF2422.

To diagnose and recover from these errors, your program should call the QMHRCVPM API again to receive the diagnostic messages sent.

Authorities and Locks

Message File Authority *USE
Message File Library Authority *USE
Message Queue Authority *CHANGE

Required Parameter Group

Message key
INPUT; CHAR(4)
The key to the inquiry message that the reply answers. The key is assigned by the command or API that sends the message. You can obtain the key in these ways:

- By receiving an inquiry message with the Receive Message (RCVMSG) command or with the Receive Program Message (QMHRCVPM) or Receive Non-program Message (QMHRCVM) API.
- Through a break handling program. See the *CL Programmer's Guide* for more information.

Qualified message queue name
INPUT; CHAR(20)
The name of the message queue containing the inquiry message being answered, and the library in which it resides. The message queue is the one in which the inquiry—not the sender's copy—is located. The first 10 characters specify the message queue, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Reply text
INPUT; CHAR(*)
The complete text of the reply being sent. To send the default reply stored in the message description, use blanks for this parameter.

Length of reply text
INPUT; BINARY(4)
The length of the reply text, in bytes. If you use blanks in the reply-text parameter to indicate that you are using the default reply, you can specify any valid value for this parameter. Valid values are 1 through 132.

Remove message
INPUT; CHAR(10)
Whether the inquiry message and the reply are removed from the message queue after the reply is sent. Valid values are:

***NO** Keep the inquiry message and the reply.
***YES** Remove the inquiry message and the reply.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24A4 E Value for remove message not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B6 E Length of text or data not valid.
CPF2401 E Not authorized to library &1.
CPF2403 E Message queue &1 in &2 not found.
CPF2408 E Not authorized to message queue &1.
CPF2410 E Message key not found in message queue &1.
CPF2411 E Not authorized to message file &1 in &2.
CPF2420 E Reply already sent for inquiry or notify message.
CPF2422 E Reply not valid.
CPF2438 D Reply value entered is not valid.
CPF2439 D Reply value entered is not valid.
CPF2440 D Reply must be in range of &1 to &2.
CPF2442 D Reply does not meet specified relations.
CPF2466 D Reply length greater than &1.
CPF2432 E Cannot send reply to message type other than *INQ or *NOTIFY.
CPF2433 E Function not allowed for system log message queue &1.
CPF2460 E Message queue &1 could not be extended.
CPF2477 E Message queue &1 currently in use.
CPF2548 E Damage to message file &1 in &2.
CPF3CF1 E Error code parameter not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9830 E Cannot assign library &1.

Part 8. Network Management APIs

Chapter 19. Network Management APIs

Network management is the process of planning, organizing, and controlling a communications-oriented system. It provides you with the capability to manage one or more nodes from another node.

The network management APIs are grouped as follows:

- SNA management services transport APIs

  Change Mode Name (QNMCHGMN) API
  Deregister Application (QNMDRGAP) API
  End Application (QNMENDAP) API
  Receive Data (QNMRCVDT) API
  Receive Operation Completion (QNMRCVOC) API
  Register Application (QNMREGAP) API
  Retrieve Mode Name (QNMRTVMN) API
  Send Error (QNMSNDER) API
  Send Reply (QNMSNDRP) API
  Send Request (QNMSNDRQ) API
  Start Application (QNMSTRAP) API

- Alert APIs

  Generate Alert (QALGENA) API
  Retrieve Alert (QALRTVA) API
  Send Alert (QALSNDA) API

- Problem log APIs

  Filter Problem (QSXFTRPB) API
  Work with Problem (QPDWRKPB) API

The network management APIs are presented in alphabetical order in this chapter.

Overview of Network Management APIs

This section provides a brief description of each category of network management APIs.

SNA Management Services Transport APIs

Systems Network Architecture (SNA) management services transport functions are used to support the sending and receiving of management services data between systems in a SNA network. The network can include AS/400 systems, Operating System/2* and NetView* licensed programs, and other platforms that support the SNA management services architecture.

The SNA management services functions provided on the AS/400 system include:

- The transport of network management data in APPN networks
- The maintenance of node relationships for network management

The APIs allow a network management application running on one system to send data to and receive data from a network management application running on another system in an APPN network. The APIs are a callable interface that allow the application to be notified about asynchronous events, such as incoming data, by way of a notification placed on a data queue.

Some examples of IBM applications that use SNA management services transport APIs are:

- Alerts
- Problem reporting
- Remote problem analysis
- Program temporary fix (PTF) ordering

In large networks, the number of sessions needed to support the various network management applications could become burdensome without session concentration. SNA management services transport APIs reduce the number of SNA LU 6.2 sessions that would normally be used to transmit data. This support multiplexes or transmits all of the network management data from all the applications in a network node domain (network node and attached end nodes) on a single session to applications in another domain.

This means that data transmitted from an end node is always sent to its network node server first. Then, the SNA management services transport support on the network node server routes the data to its proper destination.

Using the SNA Management Services Transport APIs: SNA management services is used by two types of applications: a source application and a target application. A source application sends requests to and receives replies from a target application. Similarly, a target application receives requests from and sends replies to a source application.

For a target application to receive requests from a source application, the target application must perform the following operations:

1. Create a data queue, using the Create Data Queue (CRTDTAQ) command, to allow SNA management services transport support to indicate incoming data (the MAXLEN parameter must be 80 or greater and the SEQ parameter must be *FIFO).
2. Start an application specifying the data queue just created, using the Start Application (QNMSTRAP) API. The application can then receive a handle, which will be used by the application on all following API calls. The handle is an identifier that represents the application being worked on and is unique within a job.
3. Register an application, using the Register Application (QNMREGAP) API. This notifies SNA management services transport support that the application is able to receive requests from source applications.
4. Wait for a request to arrive using the Receive Data Queue (QRCVDTAQ) API. See the *CL Programmer's Guide* for more information on the QRCVDTAQ API.

When the request arrives, the target application receives an entry from the data queue using the QRCVDTAQ API. The target application then uses the request identifier in

the entry to receive the data (call the Receive Data (QNMRCVDT) API). The entry has the following format.

Entry Format

Offset				
Dec	Hex	Type	Value	Field
0	0	CHAR(10)	*SNAMST	Entry type
10	A	CHAR(2)	01	Operation type
			02	Operation type
12	C	BINARY(31)		Handle
16	10	CHAR(53)		Request identifier
69	45	CHAR(11)		Reserved

Field Descriptions

Entry type. Indicates that this entry was created by SNA management services transport.

Operation type. Indicates which SNA management services transport API should be called next. The value 01 indicates that incoming data has arrived and that the Receive Data (QNMRCVDT) API should be called. The value 02 indicates that a previous send operation has completed and that the Receive Operation Completion (QNMRCVOC) API should be called.

Handle. This value specifies the handle of the application associated with the data queue.

Request identifier. Identifies incoming or transmitted data. This field is used when calling Receive Data (QNMRCVDT) and Receive Operation Completion (QNMRCVOC).

Then, depending on whether or not the request requires a reply, the target application may need to send a reply to the source application. A reply is sent using the Send Reply (QNMSNDRP) API.

A single request may require more than one reply. If a request is not recognized, a single error message may be sent instead of a reply using the Send Error (QNMSNDER) API.

Also, the source application must start the application (call the QNMSTRAP API), but the source does not need to create a data queue or register itself with SNA management services transport support. The source application can send a request to the target application, using the Send Request (QNMSNDRQ) API. The QNMSNDRQ API returns a request identifier that is used to receive any associated replies.

The source application receives any expected replies (call the QNMRCVDT API) with the request identifier parameter specified as the request identifier returned when the Send Request (QNMSNDRQ) API was called.

Both the target and the source applications use the End Application (QNMENDAP) API to end the management services transport support. The target application may optionally use the Deregister Application (QNMDRGAP) API before ending. However, the QNMENDAP API automatically performs a deregister operation.

The example in Figure 19-1 shows how these SNA management services transport APIs work together.

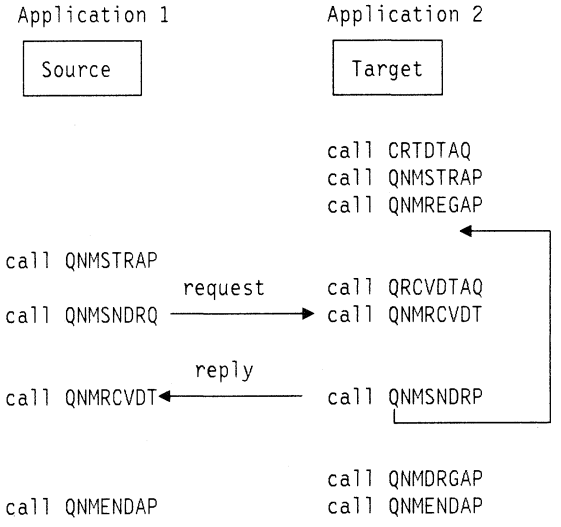


Figure 19-1. Applications Using SNA Management Services Transport APIs

Data: The types of data handled by SNA management services may be:

1. A request with a single reply expected (for example, a request for current information.
2. A request without a reply (for example, an alert)
3. A request with multiple replies expected
4. An unrecognized request returning an error message

The list correlates to the numbers on the left in Figure 19-2 on page 19-3, which shows the flow of data requests and replies depending on the parameters specified.

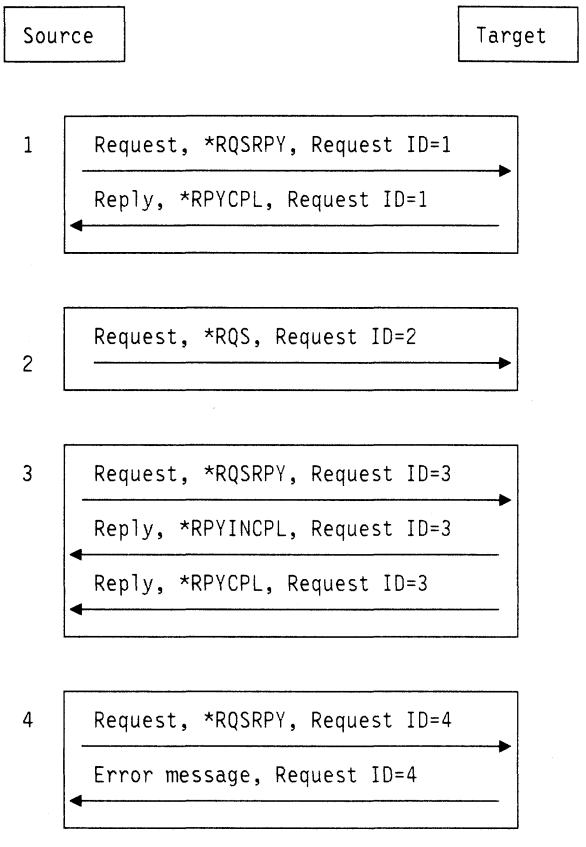


Figure 19-2. Data Types Handled by SNA Management Services Transport APIs

Notes:

1. Arrival of requests is not guaranteed unless a reply is requested. The reply acts as a form of acknowledgment.
2. The maximum amount of application data in a single request or reply is 31 739 bytes.

Routing: SNA management services transport performs all routing based on the network ID, control point name (or a logical unit name may be used), and application name. Applications must be registered with SNA management services to receive unsolicited requests.

Communications from a network node to an end node is normally performed through the end node's network node server. In the network node server, SNA management services transport provides intermediate routing functions between the end node and the network node. This is separate from the intermediate routing provided by APPN network node services.

By default, data sent using the SNA management services transport APIs uses sessions associated with system mode names CPSVCMG and SNASVCMG. CPSVCMG sessions are used between an end node and its network node server. SNASVCMG sessions are used between network nodes.

When data is sent from an end node to a network node that is not its network node server, its network node server performs intermediate routing between the CPSVCMG session and the SNASVCMG session.

Other sessions can be established by specifying a different mode name with the Change Mode Name (QNMCHGMN) API. When you change the mode name, APPN network node services performs the intermediate routing function rather than SNA management services transport.

Alert APIs

The alert APIs let your applications create alerts and notify the OS/400 alert manager of alerts that need to be handled, and allow you to retrieve alerts and alert data. These APIs differ from ordinary AS/400 system alert processing in that they let you create an alert at any time without sending an alert table message to the QSYSOPR or QHST message queue.

For more information on creating and sending OS/400 alerts, see the *Alerts and DSNX Guide*. For details about the contents of alerts built by the OS/400 system, see the *SNA Formats* manual.

Problem Log APIs

The **problem** log is a record of problems and their corresponding problem analysis status. It is used as a tool for coordinating and tracking all problem management operations. The problem log provides most of the operations necessary for problem management in a network environment.

The problem log APIs allow a user application to perform some of the operations that the problem log supports:

- Apply the active filter
- Analyze a problem
- Prepare to report a problem

Filtering: A **filter** categorizes problem log entries into groups and performs operations on them accordingly.

The problem log applies the currently active filter to a problem log entry whenever a problem entry is created, changed, or deleted using system-provided interfaces.

The operations supported allow you to send application notification to a user data queue and assign the problem to a user. Your application can receive these notifications from the data queue using existing APIs.

Work with Problem: Problem analysis is the process of finding the cause of a problem and identifying why the system is not working. Often this process identifies equipment or data communications functions as the source of the problem. The Work with Problem (QPDWRKPB) API allows you to perform problem analysis on local machine-detected problems in the problem log.

Problem reporting allows the user to prepare and report the problem, using electronic customer support for IBM support or another service provider. The Work with

Problem (QPDWRKPB) API prepares the problem in the problem log for reporting; it does not report the problem.

Change Mode Name (QNMCHGMN) API

Parameters

Required Parameter Group:

1	Handle	Input	BINARY(4)
2	Mode name	Input	Char(8)
3	Error code	I/O	Char(*)

The Change Mode Name (QNMCHGMN) API, an SNA management services transport API, sets the APPC mode name used when sending requests.

When a specific mode name is specified, direct routing is used. The result is a direct session from the source system to the target system.

Data sent using the system mode name SNASVCMG is sent at network priority, which means that data specifying this mode flows before any other data. Setting the mode to something other than SNASVCMG can lessen the effect on network performance and allow bulk data transfer. If large amounts of data are to be sent relative to the speed of the communications medium, then another mode should be considered. A mode object specified by its name in the mode name parameter must exist at the time this API is called.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Mode name
INPUT; CHAR(8)
The APPC mode name used on subsequent requests. Special values are:

***NETATR** The current default mode name specified in the network attributes.

***DFTRTG** A CPSVCMG session is used between end nodes and network nodes, and an SNASVCMG session is used between network nodes. CPSVCMG may not be specified. *DFTRTG is the default value, and it may require multiple APPN sessions to transport the data. In this case, the network node server performs SNA management services routing for all end node traffic. This results in fewer sessions in the network and a slower response time.

Note: Some non-AS/400 products only support *DFTRTG.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7AE2 E Handle &1 not found.
CPF7AE4 E Mode name &3 not valid.

Deregister Application (QNMDRGAP) API

Parameters

Required Parameter Group:

1	Handle	Input	Binary(4)
2	Error code	I/O	Char(*)

The Deregister Application (QNMDRGAP) API, an SNA management services transport API, removes the registration of the application name associated with a handle. The QNMDRGAP API only deregisters the application; the application can continue to send requests as long as replies are expected.

Only registered applications can receive requests from remote systems. After an application is deregistered, the remote applications are sent error messages notifying them that the application is not available to receive requests. In addition, it can no longer receive error messages from requests that did not request a reply. Therefore, the application should not send any request-only data.

The QNMDRGAP API is used by applications that do not want to continue receiving requests. The End Application (QNMENDAP) API automatically removes registration before ending the application instance. See the "End Application (QNMENDAP) API" on page 19-5 for more information.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7ADC E Internal processing error.

CPF7AEE E Application &2 not registered.
CPF7AE2 E Handle &1 not found.

End Application (QNMENDAP) API

Parameters

Required Parameter Group:

1	Handle	Input	Binary(4)
2	Error code	I/O	Char(*)

The End Application (QNMENDAP) API, an SNA management services transport API, ends support for the application associated with the handle. If the local application name is currently registered, its registration is automatically removed.

Applications should ensure that this API is performed when a handle is no longer needed. If you do not use QNMENDAP, the application automatically ends when the job is complete.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7AE2 E Handle &1 not found.

Filter Problem (QSXFTRPB) API

Parameters

Required Parameter Group:

1	Problem log identifier	Input	Char(30)
2	Error code	I/O	Char(*)

The Filter Problem (QSXFTRPB) API, a problem log API, applies the currently active problem log filter to a problem log entry.

The system value QPRBFTR identifies the active filter currently being used. Multiple filters can be defined, but only one can be active at a time. The QSXFTRPB API can be used at any time.

Required Parameter Group

Problem log identifier
INPUT; CHAR(30)
The problem to be retrieved, updated, and sent through the active filter. The problem log identifier has two parts: a problem ID number and the origin system. The format is:

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Problem ID number. The number the system generates to identify a problem.
10	A	CHAR(20)	Origin system. The node name of the origin system (the format is *network ID.control point name*).

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF7AA7 E Problem &1 not found.
CPF7A82 E Error occurred while applying the problem filter.
CPF7A83 E Problem filter &1/&2 not found.
CPF7A93 E Problem &2 currently in use by job &1.

Generate Alert (QALGENA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Bytes returned	Output	Binary(4)
4	Alert table (message file) and library	Input	Char(20)
5	Message ID	Input	Char(7)
6	Message data	Input	Char(*)
7	Message data length	Input	Binary(4)
8	Error code	I/O	Char(*)

The Generate Alert (QALGENA) API, an alert API, creates an alert for a particular message ID. The alert is then returned to the calling program.

Authorities and Locks

Alert Table Authority *USE
Message File Authority *READ
Library Authority *USE

Required Parameter Group

Receiver variable (alert major vector)
OUTPUT; CHAR(*)
The variable that receives the generated alert, in the format of an SNA alert major vector (for details about the format, see the *SNA Formats* manual). This area must be large enough to hold the alert. If the area is too small, the alert is not returned. A suggested size is 512 bytes because alerts cannot exceed 512 bytes for the OS/400 licensed program.

Receiver variable length
INPUT; BINARY(4)
The length of the receiver variable parameter. If this is too small, no data is returned, and the API ends with an exception.

Bytes returned
OUTPUT; BINARY(4)
If the alert is created successfully and fits in the space provided by the receiver variable parameter, this field contains the number of bytes returned.

If the alert is created successfully but cannot fit in the space provided by the receiver variable parameter, this field contains the number of bytes required to hold the created alert. The alert is not returned to the caller; error CPF7B01 is returned to the caller.

Alert table (message file) and library
INPUT; CHAR(20)
The name of the alert table where the alert description defining the alert is stored, and the name of the library in which it resides. This parameter also identifies the name of the message file to use because the alert table and message file must have the same name.

The first 10 characters contain the alert table name, and the second 10 characters contain the name of the library. Valid values for the library name are:

***CURLIB**	The job's current library. The alert table and message file must both be in this library.
***LIBL**	The library list. The alert table and message file can be in different libraries, but the libraries must both be in the library list.
specific library	The alert table and message file must be in the same library.

Message ID
INPUT; CHAR(7)
The message ID of the alert description defining the alert. This parameter also identifies the message ID of the corresponding message description. The message does not need to be an alertable message. The ALROPT parameter in the message description is ignored and an alert is always returned, provided that an alert description with the given message ID exists in the given alert table.

Message data
INPUT; CHAR(*)
The message data to use for message substitution on the alert description and message description. The format of the message data is the same as used for the Send Program Message (SNDPGMMSG) command. See the *CL Programmer's Guide* for more information about message data in general and the *CL Reference* for details about the SNDPGMMSG command.

Message data length
INPUT; BINARY(4)

The length of the message data provided by the message data parameter above.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Handling

Some the QALGENA API errors allow an alert to be returned to the application.

An alert is returned to the application in these cases:

- The user is not authorized to the message file.
- A message cannot be added to the alert because the message or message file cannot be found.

No alert is returned to the application in these cases:

- The user is not authorized to the alert table.
- OS/400 alert support cannot build an alert because no message ID is given, the alert table is missing or damaged, or no alert description is found. In this case, the alert generated could only be a cause-undetermined alert of little or no value.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7B01 E Receiver variable too small to hold alert.
CPF7B02 E Message ID not valid.
CPF7B03 E Alert table &2/&1 not found.
CPF7B04 E Alert description &1 not found in alert table &4/&3.
CPF7B05 E Message file &2/&1 not available for alert processing.
CPF7B06 E Message &1 not found in message file &4/&3 for alert processing.
CPF7B10 E Length parameter &1 is not valid.
CPF9802 E Not authorized to object &2 in &3.

Parameters

Required Parameter Group:

1	Handle	Input	Binary(4)
2	Receiver variable	Output	Char(*)
3	Receiver variable length	Input	Binary(4)
4	Request identifier	Input	Char(53)
5	Remote application name	Output	Char(24)
6	Data type	Output	Char(10)
7	Wait time	Input	Binary(4)
8	Error code	I/O	Char(*)

The Receive Data (QNMRCVDT) API, an SNA management services transport API, receives a particular request, reply, or error message.

The receiver variable data area is not changed unless it is large enough to hold all of the data received. If the receiver variable is too small to hold the received data, then the application can allocate a new buffer size greater than or equal to the bytes available and call the Receive Data (QNMRCVDT) API again. If the received data is smaller than the receiver variable, any data in the unused portion of the variable is unchanged.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Receiver variable
OUTPUT; CHAR(*)
The variable in which this API returns the data. This structure includes the bytes returned and bytes available in addition to the received data.

The format of the receiver variable is:

Offset	Type	Field
0	BINARY(4)	Bytes returned
4	BINARY(4)	Bytes available
8	CHAR(*)	Received data

The bytes returned field specifies the length of the bytes returned and the number bytes available.

Receiver variable length
INPUT; BINARY(4)
The size of the receiver variable parameter, which is the maximum amount of data that can be returned in the receiver variable.

Request identifier
INPUT; CHAR(53)
The request identifier of the data being received.

***PRV** The last request identifier used (for example, the one returned on the Send Request (QNMSNDRQ) API).

Remote application name
OUTPUT; CHAR(24)
The name of the remote application that sent the data. The first 8 characters contain the network ID, the second 8 characters contain the control point name (or the logical unit name may be used), and the third 8 characters contain the application name of the remote application.

Data type
OUTPUT; CHAR(10)
The type of data returned.

***RQS** A request was received. No reply is expected.
***RQSRPY** A request was received. A reply is expected.
***RPYCPL** A complete reply was received. This is either the last or only reply.
***RPYINCPL** An incomplete reply was received. Additional Receive Data (QNMRCVDT) operations should be performed.
***NODATA** No data was received. Either an error occurred or the wait time elapsed.

Wait time
INPUT; BINARY(4)
The amount of time the application waits for the data to be received.

-1 Waits for all the data to be received or for a condition such that the reply cannot be received (for example, a communications failure).
0-99999 The number of seconds the application waits.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF7AC0 E &3.&4 cannot receive data at this time.
CPF7AC4 E Control point &3.&4 rejected data.
CPF7ADC E Internal processing error.
CPF7ADD E Operation did not complete.
CPF7ADF E Session failure. Cannot send data at this time.
CPF7AEC E Wait time &3 not between -1 and 99999.
CPF7AE0 E Function requested not supported by remote system.
CPF7AE2 E Handle &1 not found.
CPF7AE6 E Communication with control point &3.&4 failed.
CPF7AE7 E Remote application program &5 not found.
CPF7AE8 E Receiver variable too small.
CPF7AE9 E Request identifier not valid.

Receive Operation Completion (QNMRCVOC) API

Parameters

Required Parameter Group:

1	Handle	Input	Binary(4)
2	Request identifier	Input	Char(53)
3	Remote application name	Output	Char(24)
4	Error code	I/O	Char(*)

The Receive Operation Completion (QNMRCVOC) API, an SNA management services transport API, receives an operation completion, which allows an application to determine if a send operation completed successfully. This is only used by applications that do not wait for a send operation to complete (wait time parameter equals 0). This API must be used after an application receives an entry on the data queue confirming the completion of an operation.

If the operation did not complete successfully, an error code is returned or an exception is signaled based on AS/400 error-handling rules.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Request identifier
INPUT; CHAR(53)
The request identifier of the operation completion reply that is to be received.

***PRV** The last request identifier used (for example, the one returned on the Send Request (QNMSNDRQ) API).

Remote application name
OUTPUT; CHAR(24)
The name of the remote application that received the data. The first 8 characters contain the network ID, the second 8 characters contain the control point name (or the logical unit name may be used), and the third 8 characters contain the application name of the remote application.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7AC0 E &3.&4 cannot receive data at this time.
CPF7AC2 E Previous send operation not complete.
CPF7ADC E Internal processing error.
CPF7ADF E Session failure. Cannot send data at this time.
CPF7AE0 E Function requested not supported by remote system.
CPF7AE2 E Handle &1 not found.
CPF7AE5 E Control point &3.&4 not found.
CPF7AE6 E Communication with control point &3.&4 failed.
CPF7AE7 E Remote application program &5 not found.
CPF7AE9 E Request identifier not valid.

Register Application (QNMREGAP) API

Parameters

Required Parameter Group:

1	Handle	Input	Binary(4)
2	Category	Input	Char(8)
3	Application type	Input	Char(10)
4	Replace Registration	Input	Char(10)
5	Error code	I/O	Char(*)

The Register Application (QNMREGAP) API, an SNA management services transport API, registers the application name specified in the previous start application operation so that it may receive unsolicited requests. A given application name may only be registered once on the system.

Applications must be registered to receive requests and error messages when a reply is not expected. The application name used is specified on the Start Application (QNMSTRAP) API. See the "Start Application (QNMSTRAP) API" on page 19-14 for more information.

The only applications that do not need to register are those issuing requests and expecting replies.

If the value of the replace registration parameter is *YES, a previously registered application with the same name can no longer receive requests.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Category
INPUT; CHAR(8)
The SNA management services function set group with which the application is associated.

***NONE** Not associated with any category.

Application type
INPUT; CHAR(10)
The role the application is performing:

***EPAPP** An entry point application.
***FPAPP** A focal point application.

Replace registration
INPUT; CHAR(10)
Whether or not this registration should replace a previous registration that has the same local application name.

***YES** This registration should replace any previous registration with the same local application name. Any other SNA management services program that previously registered with the same local application name can no longer receive incoming requests.

***NO** Do not replace previous registrations. If another SNA management services program in the same or different job is already registered for this local application name, then it continues to receive incoming requests. Error message CPF7AED is issued if that application is already registered.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7ADB E Replace registration value &3 not valid.
CPF7ADC E Internal processing error.
CPF7AEB E Category value &3 not valid.
CPF7AED E Application &2 already registered.
CPF7AEF E Application type &3 not valid.
CPF7AE2 E Handle &1 not found.

Retrieve Alert (QALRTVA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Format	Input	Char(8)
4	Alert log identifier	Input	Char(8)
5	Error code	I/O	Char(*)

The Retrieve Alert (QALRTVA) API, an alert API, retrieves an alert from the alert database. Different formats of data can be returned depending on what value is specified for the format parameter.

This API can be used to automate alerts such that an alert notification is sent to a data queue monitored by a user application whenever an alert is processed or received.

Required Parameter Group

Receive area
OUTPUT; CHAR(*)
The area to receive the formatted data. The data is formatted based on the format parameter. The data is either part of the alert or the alert major vector itself.

Length of receive area
INPUT; BINARY(4)
The length of the receive area. The valid lengths are from 8 bytes to *n* bytes. There is no maximum size. If enough room is not provided, then only those fields that fit are returned.

Format
INPUT; CHAR(8)
The format of the data to return. The possible formats are:

ALRT0100 Does not return the alert. It returns information about the alert that can be seen on the Work with Alerts display. See "ALRT0100 Format" for more information.

ALRT0200 Does return the alert and some information about the alert that is not contained within the alert. See "ALRT0200 Format" on page 19-10 for more information.

Alert log identifier
INPUT; CHAR(8)
The identifier used to retrieve the alert from the alert log. The log ID can be found in the alert notification record. The notification record is placed on a data queue during alert filtering by the send to data queue action.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

ALRT0100 Format

See "Field Descriptions" on page 19-10 for descriptions of the fields in this format.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	CHAR(8)	Time stamp
16	10	CHAR(10)	User assigned to this alert
26	1A	CHAR(10)	Group assigned to this alert
36	24	CHAR(20)	Filter used
56	38	CHAR(4)	Alert ID
60	3C	CHAR(10)	Problem ID
70	46	CHAR(20)	Origin system of problem
90	5A	CHAR(1)	Alert holding flag
91	5B	CHAR(1)	Local or received alert flag
92	5C	CHAR(1)	Alert held flag
93	5D	CHAR(1)	Delayed alert flag
94	5E	CHAR(1)	Operator-generated alert flag

Offset		Type	Field
Dec	Hex		
95	5F	CHAR(1)	Analysis available flag
96	60	CHAR(2)	Alert type code point
98	62	CHAR(4)	Alert description code point
102	66	CHAR(4)	First probable cause code point
106	6A	CHAR(10)	Resource name
116	74	CHAR(3)	Resource type

ALRT0200 Format

See "Field Descriptions" for descriptions of the fields in this format.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	CHAR(8)	Time stamp
16	10	CHAR(10)	User assigned to this alert
26	1A	CHAR(10)	Group assigned to this alert
36	24	CHAR(20)	Filter used
56	38	CHAR(512)	Alert major vector

Field Descriptions

Alert description code point. The description of the alert. The text is found in the QALRMSG message file in the QSYS library. The prefix for the message ID is ALD, and the suffix is the value of this field.

Alert held flag. If the alert has ever been held for the purpose of sending to the focal point, this flag is set to 1; if the alert has never been held, it is set to 0.

Alert holding flag. If the alert is currently being held to send to the focal point system, this flag is set to H; if not, the field is blank.

Alert ID. An assigned value for a particular alert.

Alert major vector. This is the SNA alert major vector.

Alert type code point. The type of alert. The text for the code point is found in the QALRMSG message file in the QSYS library. The prefix for the message ID is ALT, and the suffix is the value of this field followed by 00.

Analysis available flag. If further problem analysis is available for this problem or if the alert is for a problem analysis message, then this flag is set to 1; if the message is not for problem analysis, it is set to 0.

Bytes available. The number of bytes of data available to be returned.

Bytes returned. The number of bytes of data that were returned.

Delayed alert flag. If the alert was ever delayed, this flag is set to 1; if it has never been delayed, it is set to 0.

Filter used. The filter used to categorize the alert when it was first processed. The filter is not dynamically updated.

First probable cause code point. The most probable cause for an alert. The text for the description is found in the QALRMSG message file in the QSYS library. The prefix for the message ID is ALP, and the suffix is the value of this field.

Group assigned to this alert. The group the alert is assigned to when the alert is first filtered in the system. This value can be changed from the Work with Alerts display.

Local or received alert flag. If the alert is a locally generated alert, this flag is set to L; if it is a received alert, the flag is set to R.

Operator-generated alert flag. If the alert was generated by an operator, this flag is set to 1; if not, it is set to 0.

Origin system of problem. The system that was the origin of the associated problem entry. If no problem log entry is associated with the alert, this field is blank.

Problem ID. The ID of the problem associated with the alert. If no problem log entry is associated with the alert, this field is blank.

Resource name. The name of the resource that detected the error condition. The resource name indicates the location of the actual resource with the problem that created the alert. Generally, this is the control point name of the origin system.

Resource type. The type of resource that detected the error condition, for example, diskette, tape, printer, line, or display. The failing resource is the lowest resource in the resource hierarchy.

Time stamp. The time the alert was logged.

User assigned to this alert. The user the alert is assigned to when the alert is first filtered into the system. This value can be changed from the Work with Alerts display.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF7B10 E Length parameter &1 is not valid.
CPF7B11 E Alert not found.

Retrieve Mode Name (QNMRTVMN) API

Parameters

Required Parameter Group:

1	Handle	Input	Binary(4)
2	Mode name	Output	Char(8)
3	Error code	I/O	Char(*)

The Retrieve Mode Name (QNMRTVMN) API, an SNA management services transport API, retrieves the APPC mode name currently being used for the purpose of sending requests. The mode name retrieved is the one currently associated with the application identified by the handle.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Mode name
OUTPUT; CHAR(8)
The APPC mode name that is used when a request is sent. The special value is:

***DFTRTG** A CPSVCMG session is used between end nodes and network nodes, and a SNASVCMG session is used between network nodes. *DFTRTG is the default parameter, and it may require multiple APPN sessions to transport the data. In this case, the network node server performs SNA management services routing for all end node traffic. This results in fewer total sessions in the network and a slower response time.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7AE2 E Handle &1 not found.

Send Alert (QALSNDA) API

Parameters

Required Parameter Group:

1	Alert major vector	Input	Char(*)
2	Alert major vector length	Input	Binary(4)
3	Local or received indicator	Input	Char(1)
4	Origin	Input	Char(10)
5	Error code	I/O	Char(*)

The Send Alert (QALSNDA) API, an alert API, sends an alert to the OS/400 alert manager for processing. The alert is created by calling the Generate Alert (QALGENA) API. An alert may be received by your application from another system or it can be built by other means

When the OS/400 alert manager receives the alert, it handles it like any other alert on the system. The alert function is notified of the alert, and the alert can be logged and forwarded to a focal point or central site. The alert can be treated as either a locally generated alert or a received alert. The OS/400 alert manager updates the hierarchical information of received alerts with the name of the AS/400 control point that is handling the alert (that is, the LCLCPNAME network attribute value).

Required Parameter Group

Alert major vector
INPUT; CHAR(*)
The variable that contains the alert major vector.

Alert major vector length
INPUT; BINARY(4)
The length of the alert, in bytes. Valid values are 1 through 512.

Local or received indicator
INPUT; CHAR(1)
One of these values, indicating whether the alert is handled as locally generated or received:

L Locally generated alert. This alert is listed in the output from the Work with Alerts (WRKALR) command using the display option (DSPOPT) parameter with the *LOCAL special value. The alert hierarchy is not changed to add the current system's name.

R Received alert. This alert is listed in the output from the Work with Alerts (WRKALR) command using the display options (DSPOPT) parameter with the *RCV special value. The system name is added to the processing node list. The current system's name, stored in the LCLCPNAME network attribute, is added to the alert hierarchy.

Origin
INPUT; CHAR(10)
The origin of the alert. This value is not included in the alert. It is used only in the substitution text for messages CPI7B62 (Alert received from &1) and CPI7B60 (Incorrect alert received from &1), which are sent to the QSYSOPR message queue. Thus, you

could use it for the name of the program generating a locally generated alert, or the name of the system sending a received alert.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7B07 E Alert exceeds maximum size allowed.
CPF7B08 E Alert is not valid.
CPF7B09 E Value specified for parameter &1 not valid.
CPF7B10 E Length parameter &1 is not valid.
CPF9802 E Not authorized to object &2 in &3.

Send Error (QNMSNDER) API

Parameters

Required Parameter Group:

1	Handle	Input	Binary(4)
2	Request identifier	Input	Char(53)
3	Application error code	Input	Char(10)
4	Error code	I/O	Char(*)

The Send Error (QNMSNDER) API, an SNA management services transport API, sends an SNA management services error message to the remote application. This is used by the target application if the remote system is unable to process a request because the request data is not recognized.

If the application receiving the error message is on an AS/400 system, it will receive the error message using the Receive Data (QNMRCVDT) API. This operation results in a CPF7AC4 error message being sent, and no data is returned.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Request identifier
INPUT; CHAR(53)
The identifier for the request that contained an error.

***PRV** The last request identifier used (for example, the one specified on the Receive Data (QNMRCVDT) API).

Application error code
INPUT; CHAR(10)
The type of failure specified:

***BADDATA** The application is unable to interpret the data received.
***CANCEL** The application has canceled the processing of the this request.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7AC0 E &3.&4 cannot receive data at this time.
CPF7ADC E Internal processing error.
CPF7ADD E Operation did not complete.
CPF7ADF E Session failure. Cannot send data at this time.
CPF7AD1 E Application error code value &3 not valid.
CPF7AE2 E Handle &1 not found.
CPF7AE6 E Communication with control point &3.&4 failed.
CPF7AE9 E Request identifier not valid.

Send Reply (QNMSNDRP) API

Parameters

Required Parameter Group:

1	Handle	Input	Binary(4)
2	Request Identifier	Input	Char(53)
3	Send buffer	Input	Char(*)
4	Send buffer length	Input	Binary(4)
5	Reply type	Input	Char(10)
6	Wait time	Input	Binary(4)
7	Error code	I/O	Char(*)

The Send Reply (QNMSNDRP) API, an SNA management services transport API, sends a reply to a request that was received from a source application. Single or multiple replies may be sent for a particular request. The received request must have indicated that a reply is expected.

Multiple replies are sent by calling the Send Reply (QNMSNDRP) API one or more times. To send multiple replies, the reply type parameter is set to incomplete until the last reply is sent, when it is set to complete.

If the wait time is 0, then an entry is placed on the data queue when the operation is complete. If multiple replies are sent without waiting (wait time equals 0), multiple entries are placed on the data queue.

The same mode name used with the Send Request (QNMSNDRQ) API is used for the reply. See the"Send Request (QNMSNDRQ) API" on page 19-13 for more information about sending a request.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Request identifier
INPUT; CHAR(53)
The request identifier of the corresponding request. The request identifier is returned when the original request is received.

***PRV**	The last request identifier used (for example, the one specified on the Receive Data (QNMRCVDT) API).

Send buffer
INPUT; CHAR(*)
The data being sent.

Send buffer length
INPUT; BINARY(4)
The size of the data being sent. The send buffer can range in size from 0 through 31739.

Reply type
INPUT; CHAR(10)
The type of data being sent:

***RPYCPL**	The last or only reply (the reply is complete).
***RPYINCPL**	Additional replies will be sent (the reply is incomplete).

Wait time
INPUT; BINARY(4)
The amount of time the application waits for the send operation to complete.

-1	Waits for the send operation to complete or for a condition such that the operation cannot complete (for example, a communications failure).
0	The application does not wait for the operation to complete.
1-99999	The number of seconds the application waits.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7AC0 E &3.&4 cannot receive data at this time.
CPF7AC1 E Reply cannot be sent.
CPF7ADC E Internal processing error.
CPF7ADD E Operation did not complete.
CPF7ADF E Session failure. Cannot send data at this time.
CPF7AD2 E Send buffer length value &3 not valid.
CPF7AD5 E Reply type value &3 not valid.
CPF7AEC E Wait time &3 not between -1 and 99999.
CPF7AE2 E Handle &1 not found.
CPF7AE6 E Communication with control point &3.&4 failed.
CPF7AE9 E Request identifier not valid.

Send Request (QNMSNDRQ) API

Parameters

Required Parameter Group:

1	Handle	Input	Binary(4)
2	Remote application name	Input	Char(24)
3	Request Identifier	Output	Char(53)
4	Send buffer	Input	Char(*)
5	Send buffer length	Input	Binary(4)
6	Request type	Input	Char(10)
7	Post reply	Input	Char(10)
8	Wait time	Input	Binary(4)
9	Error code	I/O	Char(*)

The Send Request (QNMSNDRQ) API, an SNA management services transport API, sends a request to a remote application. The source application program using this API can indicate if a reply should be returned. If request-only data is sent, the source application will not know if the request data was successfully delivered. To confirm delivery, the source application must request a reply.

When the Send Request (QNMSNDRQ) API operation completes, the remote application may or may not be present. When the send request is complete, the request has been sent from the local system. If a reply is expected but the remote application does not exist, an error code is returned on the subsequent receive operation.

If a reply is not expected, the post reply parameter is ignored.

If the wait time is 0, an entry is placed on the data queue when the send request completes. The application should then perform a Receive Operation Completion (QNMRCVOC) to obtain the results of the send operation. It is recommended that a minimum of 30 seconds be used for the wait time. Larger values may be required in some networks.

The current value for the mode name of the application identified by the handle determines which session is used to send the request.

Required Parameter Group

Handle
INPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Remote application name
INPUT; CHAR(24)
The name of the remote application to which the request is sent. The first 8 characters contain the network ID, the second 8 characters contain the control point name (or the logical unit name may be used), and the third 8 characters contain the application name of the remote application.

Request identifier
OUTPUT; CHAR(53)
The identifier that the system assigned to this request. This identifier must be kept by the application to receive replies to the request (if any).

Send buffer
INPUT; CHAR(*)
The data record being sent.

Send buffer length
INPUT; BINARY(4)
The size of the data record being sent. The send buffer can range in size from 0 through 31739.

Request type
INPUT; CHAR(10)
The type of data to be sent:

***RQS**	This is a request only; no reply is expected.
***RQSRPY**	A reply is expected to this request.

Post reply
INPUT; CHAR(10)
Whether entries should be put on the data queue when replies to this request arrive.

***NO**	Do not place a reply entry on a data queue.
***YES**	Place the reply entry on the data queue associated with the handle.

Wait time
INPUT; BINARY(4)
The amount of time the application waits for the send operation to complete.

-1	Waits for the send operation to complete or for a condition such that the operation cannot complete (for example, a communications failure).
0	Does not wait for the operation to complete.
1-99999	The number of seconds the application waits.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7AC0 E &3.&4 cannot receive data at this time.
CPF7ADC E Internal processing error.
CPF7ADD E Operation did not complete.
CPF7ADF E Session failure. Cannot send data at this time.
CPF7AD2 E Send buffer length value &3 not valid.
CPF7AD6 E Request type value &3 not valid.
CPF7AD9 E Post reply value &3 not valid.
CPF7AEC E Wait time &3 not between -1 and 99999.
CPF7AE0 E Function requested not supported by remote system.
CPF7AE2 E Handle &1 not found.
CPF7AE5 E Control point &3.&4 not found.
CPF7AE6 E Communication with control point &3.&4 failed.
CPF7AE7 E Remote application program &5 not found.

Start Application (QNMSTRAP) API

Parameters

Required Parameter Group:

1	Handle	Output	Binary(4)
2	Local application name	Input	Char(8)
3	Data queue	Input	Char(20)
4	Error code	I/O	Char(*)

The Start Application (QNMSTRAP) API, an SNA management services transport API, starts support for a network management application. The handle returned by this SNA management services program must be used on all subsequent API calls to identify this application. The handle is unique to a specific job.

The Start Application API must be called before any other SNA management services transport APIs.

Authorities and Locks

Data Queue *CHANGE
Target Data Queue Other than *NONE

Required Parameter Group

Handle
OUTPUT; BINARY(4)
A variable that represents an instance of an application using some function.

Local application name
INPUT; CHAR(8)
The application name to be associated with the handle.

Data queue
INPUT; CHAR(20)
The data queue that indicates when asynchronous events occur, and enables an application to receive requests. The first 10 characters specify the data queue object name, and the last 10 characters specify the library name.

The data queue object must exist when this SNA management services program is called. The maximum entry length of the data queue must be at least 80 bytes. You must specify FIFO for the sequence of the data when you create the data queue.

***NONE**	A data queue is not used.

The following values may be specified for the library name:

***LIBL**	The library list.
***CURLIB**	The job's current library.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF7ADA E Data queue &4/&3 not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.

Work with Problem (QPDWRKPB) API

Parameters

Required Parameter Group:

1	Display panels	Input	Char (10)
2	Problem id number	Input	Char (10)
3	Origin system	Input	Char(20)
4	Current problem status	Input	Char(10)
5	Requested problem statuses	Input	Array of CHAR(10)
6	Number of requested problem statuses	Input	Bin (4)
7	Service provider network identifier	Input	Char(8)
8	Service provider control point name	Input	Char(8)
9	Problem severity	Input	Char(1)
10	Error code	I/O	Char(*)

The Work with Problem (QPDWRKPB) API, a problem log API, uses a problem log entry to analyze and prepare a machine-detected hardware problem for reporting. Only local machine-detected problems that have an OPEN, READY, PREPARED, or SENT status can be analyzed and prepared for reporting. Remote problems that have an OPEN, READY, PREPARED, or SENT status can be prepared for reporting but cannot be analyzed. This API does not analyze or prepare user-detected problems.

If a machine-detected problem is analyzed, the problem analysis program associated with the problem is called to generate a problem analysis list identifying all the possible causes for the problem.

Required Parameter Group

Displays
INPUT; CHAR(10)
Whether or not displays are shown during problem analysis. Valid values are:

***NO** No displays are shown.

Note: During problem analysis, if displays are usually shown for the type of hardware associated with the problem, then the Point of Failure list is saved. This list is only saved if no other list of causes currently associated with the problem exists.

***YES** All displays are shown. This value is not allowed if the API is called in a batch job.

Problem ID number
INPUT; CHAR(10)
The number the system generates to identify a problem.

Origin system
INPUT; CHAR(20)
The node name of the origin system (the format is *network-ID.control-point-name*).

Current problem status
INPUT; CHAR(10)
The current status of the problem. If the problem is found to be in a status other than what is indicated, an error occurs. Valid values are:

***OPENED** The problem was identified and a problem record was created.
***READY** Problem analysis information has been added to the problem record.
***PREPARED** The problem has been prepared for reporting.
***SENT** The problem record was sent to a service provider, and the information needed to correct the problem was not returned.

Requested problem statuses
INPUT; Array of CHAR(10)
The requested status for the problem. Valid values are:

***READY** Analyze the problem. Valid for local machine-detected problems only.
***PREPARED** Prepare the problem for reporting. The service provider network identifier, service provider control point name, and problem severity parameters and the default contact database are used.

Note: If you select *READY and *PREPARED, the final status is *PREPARED.

Number of requested problem statuses
INPUT; BINARY(4)
The number of statuses entered for requested problem statuses.

Service provider network identifier
INPUT; CHAR(8)
The network identifier of the service provider system where the problem is to be sent. Valid values are:

***NETATR** The network identifier of this system. Use *NETATR if the control point name is *IBMSRV.

Service provider control point name
INPUT; CHAR(8)
The control point name of the service provider system where the problem is to be sent. Valid values are:

***IBMSRV** IBM service support. This value cannot be used if the problem has an *OPENED status unless you also have a requested problem status of *READY.
Name The control point name.

Problem severity
INPUT; CHAR(1)
The severity of the problem. Valid values are:

1 A high severity level in which there is a critical affect on operations. A severity 1 problem requires a service representative at the site, and the person solving the problem must work on the problem 24 hours a day until the problem is solved or circumvented. If the problem needs more information, patching, or the problem must be created again on the failing system, a service representative must be available to do these tasks immediately. It is not a severity 1 if these and any other tasks cannot be performed immediately.
2 A medium severity level in which you are able to use the system, but your operations are severely restricted by the problem.
3 A low severity level in which you are able to continue operations with some restrictions. The restrictions do not have a critical effect on your operations.
4 The severity level is minimal because the problem causes little or no effect to your operation, or you have found a way to circumvent the problem.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF3CF2 E Error(s) occurred during running of &1 API.
CPF7AA7 E Problem &1 not found.
CPF7A93 E Problem &2 currently in use by job &1.
CPF7A9D E Problem log object is missing.
CPF7A9C E Cannot work with the problem log at this time.
CPF8C09 E &1 not defined as a service provider.
CPF8C24 E Error occurred while processing request.
CPF9308 E Unable to complete problem analysis. Reason code &1.
CPF9310 E Problem analysis procedure was exited before it had completed.
CPF9313 E Requested procedure is not allowed.
CPF9320 E &1 to display panels is not valid.
CPF9321 E Panels cannot be displayed in a batch job.
CPF9322 E Current status &3 does not match problem status &4.
CPF9323 E Current status &1 is not valid.
CPF9324 E Problems on a remote system cannot be analyzed.
CPF9325 E Problem &1 has a status that is not allowed.
CPF9326 E Problem selected not allowed.
CPF9327 E Problem analysis procedure was exited before it had completed.
CPF9328 E Severity &1 is not valid.
CPF9329 E Requested status &1 is not valid.
CPF932A E Number of requested statuses is not valid.
CPF932B E *IBMSRV not allowed as service provider for problems in OPENED status.
CPF9845 E Error occurred while opening file &1.
CPF9846 E Error while processing file &1 in library &2.

Part 9. OfficeVision/400 APIs

Chapter 20. OfficeVision/400 APIs

The OfficeVision/400* APIs allow the user customized and additional actions with office data. These are the OfficeVision/400 APIs:

Change Office Program (QOGCHGOE) allows you to set or change the Document Handling and Document Conversion exit programs.

Control Office Services (QOCCTLOF) makes requests of office services and indicates that several office tasks will be processed.

Display Directory Panels (QOKDSPDP) allows you to use the Change Directory Information display interactively without using OfficeVision/400 administration to change directory information for OfficeVision/400 users.

Retrieve Office Programs (QOGRTVOE) allows you to retrieve program names and attributes for the current Document Handling and Document Conversion exit programs.

Change Office Program (QOGCHGOE) API

Parameters

Required Parameter Group:

1	Document Handling exit program and library name	Input	Char(20)
2	Document handling program supports mail flag	Input	Char(1)
3	Document Conversion exit program and library name	Input	Char(20)
4	Error code	I/O	Char(*)

The Change Office Program (QOGCHGOE) API allows you to set or change the Document Handling and Document Conversion exit programs. The default value for these exit programs is *IBM, which indicates that OfficeVision/400 processing should be used for the request or conversion.

Authority

You must have security administrator (*SECADM) authority to call this program.

Required Parameter Group

Document Handling exit program and library name
INPUT; CHAR(20)
The name of the program called when document editing and print requests are made.

The first 10 characters contain the program name, and the second 10 characters contain the name of the library where the program is located.

***IBM** OfficeVision/400 should process document requests. The library name should be blanks if *IBM is specified in the program name.

Document handling program supports mail flag
INPUT; CHAR(1)
If an application does not plan to handle these functions, set this flag to 0. If the document handling program is *IBM, set this value to 1.

0 Mail requests are not supported.
1 Mail requests are supported.

This indicates whether the program specified in the Document Handling exit program and library name parameter should be called for mail requests.

Document Conversion exit program and library name
INPUT; CHAR(20)
The name of the program that is used to do document conversions when office services requires the document to be in a different data stream format.

The first 10 characters contain the program name, and the second 10 characters contain the name of the library where the program is located.

***IBM** Use the IBM-supplied Document Conversion exit program. The library name should be blank if *IBM is specified in the program name.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF90A8	*SECADM special authority required to do requested operation.
OFC15FF	Value &1 for mail handled parameter is not valid.
CPF3C29	Object name &1 is not valid.

Control Office Services (QOCCTLOF) API

Parameters

Required Parameter Group:

1	Request type	Input	Char(10)
2	Error code	I/O	Char(*)

The Control Office Services (QOCCTLOF) API makes requests of the office services. Office services accepts the following actions:

- *START
- *END
- *CHECK

Required Parameter Group

Request type
INPUT; CHAR(10)
The request that the application makes of the office services. Possible values are:

***START**	Start an office services block. An office services block is the time during the running of an application when more office commands will be used. Office services leaves the job in such a state as to maximize the performance of subsequent office services use. This includes leaving the office files open and leaving working spaces created and already initialized. It is the responsibility of the application to end the office services block.
***END**	End an office services block. No further office services will be used. The files will be closed and all working spaces deleted.
***CHECK**	Determine if an office services block is active. An error will be returned if an office services block has not been previously started or has already ended.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

OFCFD01	Office services session is already active.
OFCFD02	Office services session request failed.
OFCFD03	Office services session is not active.
OFCFD04	Office services session request is not correct.
OFCFD05	Number of bytes provided for the error code is not correct.

Display Directory Panels (QOKDSPDP) API

Parameters

Required Parameter Group:

1	User ID	Input	Char(8)
2	User address	Input	Char(8)
3	Title	Input	Char(10)
4	Function key processing	Output	Char(10)
5	Message to be displayed	Input	Char(*)
6	Error code	I/O	Char(*)

The Directory Panels (QOKDSPDP) API has the ability to display the Change Directory Information display. It can run that function interactively without using OfficeVision/400 administration to change directory information for office users. This API is for interactive use only.

Authorities

If you have security administrator (*SECADM) authority, you can change all the directory information. If you do not have security administrator authority, then you can only change your own directory information.

Required Parameter Group

User ID
INPUT; CHAR(8)
The user ID to be processed. The ID needs to be in character set 697, code page 500, and monocase.

User address
INPUT; CHAR(8)
The address of the user to be processed. The address needs to be in character set 697, code page 500, and monocase.

Title
INPUT; CHAR(10)
The type of directory panel to be displayed. The values are:

***CHG**	Change directory information
***ADD**	Add directory information

Function key processing
OUTPUT; CHAR(10)
The operation processed for the key pressed. This indicates what type of ending processing to do. The values are:

***ENTER**	The enter operation is processed.
***F3**	The exit operation is processed.
***F12**	The previous display is shown.

Message to be displayed
INPUT; CHAR(*)
A structure with a message ID and replacement text that is to be shown on the display. If the message data length is 0, no message will be displayed. For the format of the structure, see "Format of Messages to Be Displayed."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of Messages to Be Displayed

The format of the message to be displayed parameter is as follows:

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Message size
4	4	CHAR(7)	Message name
11	0B	CHAR(*)	Message data

Field Descriptions

Message data. The data to be replaced in the message.

Message name. The message ID of the message to be displayed.

Message size. The length of the area that the calling application provides for the message data.

Error Messages

CPF89A0 Values passed to QOKDSPDP are not valid.
CPF9083 User ID and address &1 &2 not changed.

Retrieve Office Programs (QOGRTVOE) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Error code	I/O	Char(*)

The Retrieve Office Programs (QOGRTVOE) API retrieves the programs that are used for office document requests and for document conversion requests. The default value for the programs being retrieved is *IBM, which indicates that OfficeVision/400 processing should be used for the request or conversion.

Authority

To use this program you need *SECADM authority.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable to receive exit program information. This area must be large enough to accommodate the information specified.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. The length must be at least 8 bytes. If the variable is not large enough to hold the subsystem information, the data is truncated.

Format name
INPUT; CHAR(8)
The format of the program to be retrieved. The valid format is OGOE0100 (see "OGOE0100 Format").

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

OGOE0100 Format

The following list shows the format of the program to be retrieved. For detailed descriptions of the fields, see "Field Descriptions" on page 20-3.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Document handling program name
18	12	CHAR(10)	Document handling program library name
28	1C	CHAR(1)	Document handling program supports mail flag
29	1D	CHAR(10)	Document conversion program name
39	27	CHAR(10)	Document conversion program library name

Field Descriptions

Bytes available. The total length of all data available.

Bytes returned. The length of the data actually returned.

Document conversion program library. The name of the library containing the program to be called for document conversions. This field will be blanks if the document conversion exit program name is *IBM.

Document conversion program name. The name of the program to be called for document conversions if an IBM conversion does not exist. If the program is *IBM, the IBM-supplied conversion exit or registered conversions will be called.

Document handling program handles mail flag. An indication of whether the document handling program supports requests from the Work with Mail display (that is, MAILVIEW, MAILFWD, MAILREPLY, PRINT and PRINTOPTS). If *IBM is specified as the document handling exit program, this value will be 1.

0 The exit program does not support mail.
1 The exit program does support mail.

Document handling program library. The name of the library containing the program to be called for document requests. This field will be blanks if the document handling exit program name is *IBM.

Document handling program name. The name of the program to be called for document requests. If the program is *IBM, OfficeVision/400 is called for the document requests.

Error Messages

CPF3CF1	Error code parameter not valid.
CPF90A8	*SECADM special authority required to do requested operation.
CPF3C24	Length of the receiver variable is not valid.
CPF3C21	Format name &1 is not valid.

Chapter 21. OfficeVision/400 Exit Programs

You may write exit programs that are called by the OfficeVision/400 program to customize the OfficeVision/400 functions to your needs.

Directory Search exit program allows the administrator to define a customized search of directory data. The Directory Search program is specified on the SCHPGM parameter of the Change Directory Attribute (CHGDIRA) command.

Directory Verification exit program allows the administrator to define additional security or validation checking when directory data is added, changed, or deleted. The verification program is specified on the VRFPGM parameter of the Change Directory Attribute (CHGDIRA) command.

Document Conversion exit program allows other document conversion programs to be called when a request is made for the OfficeVision/400 program to process a document that is an unsupported type (for example, PCFILE).

Document Handling exit program allows other editors and applications to be called from the OfficeVision/400 word processing and print functions.

Directory Search Exit Program

Parameters

Required Parameter Group:

1	Function being requested	Input	Char(10)
2	Maximum entries for addressees	Input	Binary(4)
3	Maximum entries for copy list	Input	Binary(4)
4	Number of array elements	Output	Binary(4)
5	Array	Output	Char(*)

The Directory Search exit program allows the administrator to define a customized search of directory data. The exit will be one of three functions: display, select, or optional select (addressee or copy list selection). In this way, a user-customized search is provided and this search is integrated with OfficeVision/400 programs and the system distribution directory.

Use the Change Directory Attributes (CHGDIRA) command to specify the Directory Search exit program name in the SCHPGM parameter. When F10 is pressed on the Search System Directory display, the Directory Search exit program is called.

Required Parameter Group

Function being requested
INPUT; CHAR(10)
The type of search operation that was requested.

***DISPLAY**
Search and display directory information.

***SELECT**
Search and select user IDs. The output includes user IDs and addresses selected as a result of a search function.

***OPTSELECT**
Search and select user IDs for addressees and copy list.

Maximum entries for addressees
INPUT; BINARY(4)
The maximum number of entries that can be selected for the addressees of the distribution. The maximum this number can be is 100.

This parameter is used with the *SELECT and *OPTSELECT values of the function being requested parameter. If the function is *DISPLAY, this value is 0 indicating no output is returned.

Maximum entries for copy list
INPUT; BINARY(4)
The maximum number of entries that can be selected to be copied on the distribution. This is the maximum number of entries that can be returned in the array parameter with the value of 2 in select option used field. The maximum this number can be is 100.

This value is used with the *OPTSELECT value of the function being requested parameter. If the function is *DISPLAY, this value will be 0 indicating no output is to be returned.

Number of array elements
OUTPUT; BINARY(4)
For the values *SELECT and *OPTSELECT, the number of users that is returned. The maximum value is the total of the maximum entries for addressees parameter plus the maximum entries for copy list parameter.

Array
OUTPUT; CHAR(*)
The array of the user IDs and addresses selected from the search function. For the format of this character parameter, see Figure 21-1.

Figure 21-1. Array Information

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Select option used
1	1	CHAR(8)	User ID
9	9	CHAR(8)	User address
17	11	CHAR(50)	User's full name or description

Field Descriptions

Select option used. The number of the option you want to use.

| 1 Addressee list
| 2 Carbon copy list

User address. The second part of a two-part network name used in the system distribution directory and in the office applications to uniquely identify a user and to send electronic mail.

User ID. The first part of a two-part network name used in the system distribution directory and in the office applications to uniquely identify a user.

User's full name or description. The user's full name as it appears when a directory is searched or viewed. The directory can contain more than one description for the user ID and address; however, each description must be unique for the same user ID and address.

Directory Verification Exit Program

Parameters

Required Parameter Group:

1	Function being requested	Input	Char(10)
2	Directory information format	Input	Char(10)
3	Owning system name	Input	Char(8)
4	User making request	Input	Char(10)
5	System making request	Input	Char(8)
6	Length of directory information	Input	Binary(4)
7	Directory Information	Input	Char(*)

The verification program allows the administrator to define additional security or syntax checking on the data. If defined, the user exit program is called before any directory entry, department, or location is added, changed, or removed from the system. The verification program is specified on the VRFPGM parameter of the Change Directory Attribute (CHGDIRA) command.

The verification program is given all updated information known about the directory entry, department, or location. Your exit program returns to the directory service an indication as to whether the add, change, or delete operation is to be applied to the Enterprise Address Book (on the AS/400 system, this is called the system distribution directory). The EAB is a collection of data, such as information about people, departments, and locations in a network. An example of an enterprise is a company.

Whether or not update requests that are rejected by the exit program are supplied to other systems depends on the origin of the update.

- When a local, interactive update request is rejected by the exit program, the update does not affect the local system. If you are participating in directory shadowing, the update is not supplied to other systems in the directory shadowing network.
- If you are participating in directory shadowing and a shadowed update request is received and is rejected by the exit program, then the update does not affect the local system.

  Other systems in the directory shadowing network will be affected. The results depend on the nature of the shadowed update request:
- When an add request is rejected, it is filtered out of (does not appear on) the local system, but remains on the network. Information about the add request is stored separately in the change log and will subsequently be supplied to other systems in the directory shadowing network.
- When a change or delete request is rejected, the local system keeps its original information. In addition, the original information will be supplied again to other systems, to keep directory information consistent throughout the network. You should have matching verification criteria on all your systems to help reduce the amount of processing on the network.

In order for a system to get back any of the data that was filtered out, you need to do an IPL from a system that has that data.

Required Parameter Group

Function being requested
INPUT; CHAR(10)
The type of operation that the user is attempting to do to the directory information that is described by the other parameters.

***ADD**	Information is being added.
***ADDDSC**	User description is being added.
***CHG**	Information is being changed. All fields in the directory information that are not changed will be X'00' except for the first field of every table. That field indicates what directory entry, department, or location is being changed.
***DLT**	Information is being deleted.
***DLTDSC**	User description is being deleted.

Directory information format
INPUT; CHAR(10)
The format of the directory information that is being worked with. The information is provided in the directory information parameter. The valid formats are:

CHKP0100	Directory entry (see "CHKP0100 Format" on page 21-3)
CHKP0200	Department (see "CHKP0200 Format" on page 21-5)
CHKP0300	Location (see "CHKP0300 Format" on page 21-5)

Owning system name
INPUT; CHAR(8)
The name of the system that "owns" the directory entry, department, or location that is being worked with. The owning system is the system that originally added the data to the network.

***LOCAL**	The entry is owned by the local system.

User making request
INPUT; CHAR(10)
The user profile of the user that is doing the request. When using the shadowing function, this is the user that originated the modification.

System making request
INPUT; CHAR(8)
The system from where the request is coming. When using the shadowing function, this is the system that originated the modification.

Length of directory information
INPUT; BINARY(4)
The length of the directory information in the directory information parameter. The length depends on the directory information format. Each format has a different (but fixed) length as shown in specific format tables.

Directory information
INPUT; CHAR(*)
The directory information that is associated with the directory entry, department, or location that the request is made against. For the format of this character parameter, refer to the specific format table ("CHKP0100 Format," "CHKP0200 Format" on page 21-5 , or "CHKP0300 Format" on page 21-5).

Rejecting an Update Operation: The customer-written exit program may reject update requests. To do so, the exit program must signal a specific program message to the directory services module that called it, and then return. An update is rejected based on the restrictions you identify. For example, if three people have authority to make updates but you want the program to allow only one user to do updates, then update requests by all other users are rejected.

To allow a directory update request, the exit program only returns to the program that called it. Two program messages have been defined for you to use for the purpose of rejecting directory updates. The messages are:

CPF89A3 Operation not successful due to authority reasons.
CPF89A4 Operation not successful due to data validation reasons.

You may provide an optional data structure with message variable substitution text. This will help clarify the reasons for the rejection to your users. The optional data structure is:

CHAR(10) The user ID of the person who requested (entered) the update. You can get this from the user making request parameter.
CHAR(8) The system name of the person who requested (entered) the update. You can get this from the system making request parameter.
CHAR(120) A description of the reason your exit program is rejecting the request.

CHKP0100 Format

Figure 21-2 (Page 1 of 3). Directory Entry Format

Offset		Type	Field
Dec	**Hex**		
Note: The following fields are in code page 500 and character set 697.			
0	0	CHAR(16)	User ID/address
16	10	CHAR(16)	System name/group
32	20	CHAR(10)	User profile
42	2A	CHAR(47)	Network user ID
89	59	CHAR(16)	New user ID/address
105	69	CHAR(16)	Old user to forward from user ID/address
121	79	CHAR(1)	Indirect user
122	7A	CHAR(1)	Print personal mail
123	7B	CHAR(3)	Reserved
Note: The following fields are in character sets and code pages as defined by their individual fields.			
126	7E	CHAR(50)	Description
176	B0	BINARY(4)	Character set of the above field
180	B4	BINARY(4)	Code page of the above field
184	B8	CHAR(40)	Last name
224	E0	BINARY(4)	Character set of the above field
228	E4	BINARY(4)	Code page of the above field
232	E8	CHAR(20)	First name
252	FC	BINARY(4)	Character set of the above field
256	100	BINARY(4)	Code page of the above field
260	104	CHAR(20)	Middle name
280	118	BINARY(4)	Character set of the above field
284	11C	BINARY(4)	Code page of the above field
288	120	CHAR(20)	Preferred name
308	134	BINARY(4)	Character set of the above field
312	138	BINARY(4)	Code page of the above field
316	13C	CHAR(2)	Reserved
318	13E	CHAR(50)	Full name
368	170	BINARY(4)	Character set of the above field
372	174	BINARY(4)	Code page of the above field
376	178	CHAR(2)	Reserved
378	17A	CHAR(10)	Department
388	184	BINARY(4)	Character set of the above field

Figure 21-2 (Page 2 of 3). Directory Entry Format

Offset		Type	Field
Dec	Hex		
392	188	BINARY(4)	Code page of the above field
396	18C	CHAR(2)	Reserved
398	18E	CHAR(50)	Job title
448	1C0	BINARY(4)	Character set of the above field
452	1C4	BINARY(4)	Code page of the above field
456	1C8	CHAR(2)	Reserved
458	1CA	CHAR(50)	Company
508	1FC	BINARY(4)	Character set of the above field
512	200	BINARY(4)	Code page of the above field
516	204	CHAR(2)	Reserved
518	206	CHAR(26)	Telephone number 1
544	220	BINARY(4)	Character set of the above field
548	224	BINARY(4)	Code page of the above field
552	228	CHAR(2)	Reserved
554	22A	CHAR(26)	Telephone number 2
580	244	BINARY(4)	Character set of the above field
584	248	BINARY(4)	Code page of the above field
588	24C	CHAR(40)	Location
628	274	BINARY(4)	Character set of the above field
632	278	BINARY(4)	Code page of the above field
636	27C	CHAR(20)	Building
656	290	BINARY(4)	Character set of the above field
660	294	BINARY(4)	Code page of the above field
664	298	CHAR(16)	Office
680	2A8	BINARY(4)	Character set of the above field
684	2AC	BINARY(4)	Code page of the above field
688	2B0	CHAR(40)	Mailing address line 1
728	2D8	BINARY(4)	Character set of the above field
732	2DC	BINARY(4)	Code page of the above field
736	2E0	CHAR(40)	Mailing address line 2
776	308	BINARY(4)	Character set of the above field

Figure 21-2 (Page 2 of 3). Directory Entry Format

Offset		Type	Field
Dec	Hex		
780	30C	BINARY(4)	Code page of the above field
784	310	CHAR(40)	Mailing address line 3
824	338	BINARY(4)	Character set of the above field
828	33C	BINARY(4)	Code page of the above field
832	340	CHAR(40)	Mailing address line 4
872	368	BINARY(4)	Character set of the above field
876	36C	BINARY(4)	Code page of the above field
880	370	CHAR(2)	Reserved
882	372	CHAR(50)	Text
932	3A4	BINARY(4)	Character set of the above field
936	3A8	BINARY(4)	Code page of the above field
940	3AC	CHAR(1)	Print cover page
941	3AD	CHAR(1)	Mail notification
Note: The following fields are in the character set and code page as defined by 1984 X.400 standards.			
942	3AE	CHAR(3)	X.400 country
945	3B1	CHAR(16)	X.400 administration domain
961	3C1	CHAR(16)	X.400 private domain
977	3D1	CHAR(64)	X.400 organization
1041	411	CHAR(40)	X.400 surname
1081	439	CHAR(16)	X.400 given name
1097	449	CHAR(5)	X.400 initials
1102	44E	CHAR(3)	X.400 generation qualifier
1105	451	CHAR(32)	X.400 organization unit 1
1137	471	CHAR(32)	X.400 organization unit 2
1169	491	CHAR(32)	X.400 organization unit 3
1201	4B1	CHAR(32)	X.400 organization unit 4
1233	4D1	CHAR(8)	X.400 domain attribute type 1
1241	4D9	CHAR(128)	X.400 domain attribute value 1
1369	559	CHAR(8)	X.400 domain attribute type 2
1377	561	CHAR(128)	X.400 domain attribute value 2
1505	5E1	CHAR(8)	X.400 domain attribute type 3
1513	5E9	CHAR(128)	X.400 domain attribute value 3
1641	669	CHAR(8)	X.400 domain attribute type 4

Figure 21-2 (Page 3 of 3). Directory Entry Format

Offset		Type	Field
Dec	Hex		
1649	671	CHAR(128)	X.400 domain attribute value 4

Note: All fields in the directory information that are not changed will be X'00' except for the first field of every table. That field indicates what directory entry, department, or location is being changed.

CHKP0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(2)	Reserved
2	2	CHAR(10)	Department
12	C	BINARY(4)	Character set of the above field
16	10	BINARY(4)	Code page of the above field
20	14	CHAR(2)	Reserved
22	16	CHAR(50)	Title
72	48	BINARY(4)	Character set of the above field
76	4C	BINARY(4)	Code page of the above field
80	50	CHAR(2)	Reserved
82	52	CHAR(10)	Reports to department
92	5C	BINARY(4)	Character set of the above field
96	60	BINARY(4)	Code page of the above field
Note: The following field is in code page 500 and character set 697.			
100	64	CHAR(16)	Manager user ID/address
Note: All fields in the directory information that are not changed will be X'00' except for the first field of every table. That field indicates what directory entry, department, or location is being changed.			

CHKP0300 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(40)	Location
40	28	BINARY(4)	Character set of the above field
44	2C	BINARY(4)	Code page of the above field
48	30	CHAR(2)	Reserved
50	32	CHAR(30)	Location line 1
80	50	BINARY(4)	Character set of the above field
84	54	BINARY(4)	Code page of the above field
88	58	CHAR(2)	Reserved
90	5A	CHAR(30)	Location line 2
120	78	BINARY(4)	Character set of the above field
124	7C	BINARY(4)	Code page of the above field
128	80	CHAR(2)	Reserved
130	82	CHAR(30)	Location line 3
160	A0	BINARY(4)	Character set of the above field
164	A4	BINARY(4)	Code page of the above field
168	A8	CHAR(2)	Reserved
170	AA	CHAR(30)	Location line 4
200	C8	BINARY(4)	Character set of the above field
204	CC	BINARY(4)	Code page of the above field
208	D0	CHAR(2)	Reserved
210	D2	CHAR(30)	Location line 5
240	F0	BINARY(4)	Character set of the above field
244	F4	BINARY(4)	Code page of the above field
248	F8	CHAR(2)	Reserved
250	FA	CHAR(30)	Location line 6
280	118	BINARY(4)	Character set of the above field
284	11C	BINARY(4)	Code page of the above field
Note: All fields in the directory information that are not changed will be X'00' except for the first field of every table. That field indicates what directory entry, department, or location is being changed.			

Field Descriptions

Building. The name or number that identifies the user's building.

Character set of the above field. The character identifier (graphic character set) that was used by the work station to enter the data for the field.

Code page of the above field. The value specified on this parameter is used to instruct the printer device to interpret the hexadecimal byte string to print the same characters that were intended when the text was created.

Company. The name of the company for whom the user works.

Department. The name or number that identifies the user's department.

Description. The description associated with the user ID. One entry in the directory can have several different descriptions.

First name. The user's first name or given name.

Full name. The user's full name as it appears when a directory is viewed or searched.

Indirect user. A person enrolled in the system distribution directory who receives mail but never signs on to view it. An indirect user receives printed mail only. The values are 0 for no and 1 for yes.

Job title. The title of the user's occupation.

Last name. The user's last name.

Location. The location of the business or system. Some examples of location are city, state, or street address. Use the Work with Directory Locations command (WRKDIRLOC) to see the locations defined.

Location lines 1 through 6. The location of your business or system. These fields further describe a location name. For example, the field may contain the general mailing address for the location.

Mail notification. Whether or not you want to be notified when mail arrives.

These are the values you can specify:

Blank	Notified for priority or personal mail and messages
1	Notified for priority or personal mail and messages
2	Notified for priority or personal mail only
3	Notified for messages only
4	Notified for all mail
0	Notified for no mail

Mailing address lines 1 through 4. The address of the user. You can use up to four lines for the address.

Manager user ID/address. The department manager's user ID and address.

Middle name. The user's middle name.

Network user ID. A unique value associated with each user in the Enterprise Address Book. For example, the value could be the user ID/address, the social security number, or the employee number. This field is filled in for all operations using format CHKP0100.

New user ID/address. The new user ID and address used during the rename operation.

Office. The name or number that identifies the user's office.

Old user to forward from user ID/address. This field shows the previous user ID and address from which you want to forward mail.

Preferred name. The name by which the user prefers to be known.

Print cover page. Whether a cover page is printed when your mail is printed. The values are 0 for no and 1 for yes.

Print personal mail. This field is used only if the user is an indirect user. The value specifies whether to print the mail that can be accessed only by the receiver, but not by someone working on behalf of the receiver. When mail is sent, it can be assigned the classification personal. The values are 0 for no and 1 for yes.

Reports to department. The department to which this department reports.

Reserved. An ignored field.

System name/group. An IBM-supplied name that uniquely identifies the system. It is used as a network value for certain communications applications such as APPC.

Telephone number 1. The telephone number of the user's office or business, or telephone numbers that are significant to the user. The most important number should be listed on the first line because only the first line is displayed when you use the search directory function.

Telephone number 2. The second line for telephone numbers.

Text. Any additional information that describes the entry.

Title. A department title that further describes the department name.

User ID/address. A two-part network name used in the system distribution directory and in the office applications to uniquely identify a user and to send electronic mail. This field is filled in for all operations using format CHKP0100.

User profile. The user profile, if any, associated with a user ID and address.

X.400 administration domain. The administration management domain part of the X.400 originator/recipient (O/R) name. An administration management domain is a management domain that is administered by a public organization, such as a national Post Telephone and Telegraph Administration (PTT). A management domain is a set of message transfer agents (MTAs) and user agents (UAs) that comprise a message handling system.

X.400 country. The country part of the X.400 originator/recipient (O/R) name.

X.400 domain attribute types 1 through 4. The type of a domain-defined attribute for this object. The domain-defined attribute is not defined by X.400 standards but is allowed in the X.400 originator/recipient (O/R) name to accommodate values of existing systems sending messages.

X.400 domain attribute values 1 through 4. The code immediately following the attribute type that specifies a particular property from the set defined by the attribute type.

X.400 generation qualifier. The generation qualifier part of the X.400 originator/recipient (O/R) name. For example, the generation qualifier in the name John R. Smith, III is III. If you specify a generation qualifier, you must specify an X.400 surname.

X.400 given name. The user first name, or given name, part of the X.400 originator/recipient (O/R) name. The default for the given name is the equivalent of the first name. If you specify a given name, you must specify an X.400 surname.

X.400 initials. The first and middle initials of the X.400 originator/recipient (O/R) name. For example, the initials for John Henry Smith are JH. If you specify initials, you must specify a surname.

X.400 organization. The organization name part of the X.400 originator/recipient (O/R) name.

X.400 organization units 1 through 4. The organization-defined unit part of the X.400 originator/recipient (O/R) name.

X.400 private domain. The private management domain part of the X.400 originator/recipient (O/R) name. A private management domain is a management domain that is administered by a private company or a noncommercial organization.

X.400 surname. The user last name, or surname, part of the X.400 originator/recipient (O/R) name.

Error Messages

CPF89A3	Operation not successful due to authority reasons.
CPF89A4	Operation not successful due to data validation reasons.

Document Conversion Exit Program

Parameters

Required Parameter Group:

1	Input document name	Input	Char(12)
2	Input folder name	Input	Char(63)
3	Input document type	Input	Binary(4)
4	Output document name	Input	Char(12)
5	Output folder name	Input	Char(63)
6	Output document type	Input	Binary(4)
7	Function code	Input	Char(1)
8	Conversion existence indicator	Output	Char(1)

The Document Conversion exit program allows other document conversions to be called when a request is made for the OfficeVision/400 program to process a document type that it does not support. The OS/400 and OfficeVision/400 programs use document conversions when opening documents.

Required Parameter Group

Input document name
INPUT; CHAR(12)
The name of the document the function is to be performed against.

Input folder name
INPUT; CHAR(63)
The folder in which the document is to be found.

Input document type
INPUT; BINARY(4)
The type of the document that is being worked with. The value must be in the range of 1 through 65535.

Output document name
INPUT; CHAR(12)
The name of the output document.

Output folder name
INPUT; CHAR(63)
The folder in which the output document is to be placed.

Output document type
INPUT; BINARY(4)
The format of the document that is being worked with. The value must be in the range of 1 through 65535.

Function code
INPUT; CHAR(1)
Whether the exit is being called to check for the existence of a conversion or to perform the conversion.

- **0** A conversion is being requested.
- **1** An existence check is requested.

Conversion existence indicator
OUTPUT; CHAR(1)
Whether the requested conversion exists. This flag must be set for both conversion existence checks and conversion requests.

- **0** Conversion does not exist.
- **1** Conversion exists.

Use of Document Conversions by IBM Programs:

Other document conversions will be called when you are opening documents and:

- Processing the following document commands and using the OfficeVision/400 editor or print functions:
  - CRTDOC
  - EDTDOC
  - DSPDOC
  - PRTDOC
  - CHKDOC
  - MRGDOC
  - PAGDOC
  - ADDTXTIDXE
- Processing the following options from the Work with Documents in Folders display and using OfficeVision/400:
  - Create
  - Revise
  - View
  - Print
  - Spell
  - Paginate
  - Print options
- Processing the following options from the Work with Documents in a Document List display:
  - Revise
  - View
  - Print
- Processing the following options from the Work with Mail display and using OfficeVision/400:
  - Revise a copy
  - View
  - Print
  - Forward
  - Reply
- Copying a document on a create request when using the OfficeVision/400 editor.
- Recalling from or copying or moving to a notepad document within the OfficeVision/400 editor.
- Using the GET function within the OfficeVision/400 editor.
- Including a document with the .inc instruction in the OfficeVision/400 print function.
- Accessing external footnote documents in the OfficeVision/400 print function.
- Accessing a fill-in document (merge from a document) in the OfficeVision/400 print function.
- Accessing shell documents QPROFNOT and QPROFDOC for converting incoming PROFS* documents and notes.
- Printing mail for indirect office users.
- Accessing the shell note when creating a note.
- Accessing the output document for a copy graph or image request.

In each of these cases a conversion from the current format to RFTDCA or FFTDCA is requested. A second IBM conversion from DCA* to AS/400 format may also be requested using the IBM conversion programs.

Document Handling Exit Program

Parameters

Required Parameter Group:

1	Document name	Input	Char(12)
2	Folder name	Input	Char(63)
3	Document type	Input	Binary(4)
4	Function	Input	Char(10)
5	Function-specific information	Input	Char(485)
6	Exit processing indicator	Output	Char(4)

The Document Handling program allows other applications to be called in place of or in addition to the OfficeVision/400 word processor. Requests are sent to the exit program. The exit program can choose to process the request or return it to the OfficeVision/400 program for processing.

Required Parameter Group

Document name
INPUT; CHAR(12)
The name of the document on which the function is performed.

Folder name
INPUT; CHAR(63)
The folder in which the document is located.

Document type
INPUT; BINARY(4)
The format of the document that is being worked with. The value must be from 1 to 65535.

Function
INPUT; CHAR(10)
The type of operation that the user is requesting for this document.

Refer to the "DOCI0100 Format" on page 21-10 for print function requests.

Refer to the "DOCI0200 Format" on page 21-11 for merge function requests.

Refer to the "DOCI0300 Format" on page 21-11 for spell function requests.

Refer to the "DOCI0400 Format" on page 21-11 for mail function requests.

Refer to the "DOCI0500 Format" on page 21-11 for edit function requests.

Refer to the "DOCI0600 Format" on page 21-11 for create function requests.

Function-specific information
INPUT; CHAR(485)
Additional input that varies by function. See the Figure 21-3 on page 21-9 for more detailed information.

Note: Function-specific information that is not specified and for which there is no value stored in the document will be passed to the exit program as blanks.

Exit processing indicator
OUTPUT; CHAR(4)
The additional processing that the IBM programs should perform on return from the document handling program.

0000	No additional processing required for this request. (Processed like the enter key from the exit.)
0001	Request has been processed, and the user requested to return. The IBM programs process as if F12 were pressed.
0002	Request has been processed, and the user requested to return. The IBM programs process as if F3 were pressed.
0007	Document has been deleted (only returned on VIEWMAIL requests).
0010	The OfficeVision/400 program should process the request.

Figure 21-3 shows when the exit program will be called for each of the document functions.

Figure 21-3 (Page 1 of 2). Document Handling Function Requests

Function	Description	Where Called From
'CREATE'	Create document (DOCI0600 format)	• CRTDOC command • *Work with Documents in Folders* display: option 1 **Note**: The exit program is called after the Enter key or F10 is pressed on the Create Document Details display, when details are used.
'VIEW'	View document (DOCI0100 format)	• DSPDOC command • MERGE request with DSPOPT(*VIEW) • *Work with Documents in Folders* display; option 5 • *Work with Documents in a List* display; option 5
'EDIT'	Edit document and revise a copy of a mail item (DOCI0500 format)	• EDTDOC command • MERGE request with DSPOPT(*EDIT) • *Work with Documents in Folders* display: option 2 • *Work with Documents in a List* display; option 2 • *Work with Mail* display; option 2
'MAILVIEW'	View mail item (DOCI0400 format)	• *Work with Mail* display; option 5

Figure 21-3 (Page 1 of 2). Document Handling Function Requests

Function	Description	Where Called From
'MAILEDIT'	Create a note (DOCI0400 format)	• *OfficeVision/400* main menu; option 4 **Note**: The exit program is called after F6 (Type Note) is pressed on the *Send Note* display.
'MAILFWD'	Forward a note (DOCI0400 format)	• *Work with Mail* display; option 10 **Note**: The exit program is called after F6 (Type Note) is pressed on the *Forward Mail* display.
'MAILREPLY'	Reply to a note (DOCI0400 format)	• *Work with Mail* display; option 11 **Note**: The exit program is called after F6 (Type Note) is pressed on the *Reply to Mail* display.
'MERGE'	Merge document (DOCI0200 format)	• MRGDOC command; display merge options is *NO
'MERGEOPTS'	Merge document with options (DOCI0200 format)	• MRGDOC command; display merge options is *YES
'PRINT'	Print request (DOCI0100 format)	• PRTDOC command; display print options is *NO • *Work with Documents in Folders* display; option 6 • *Work with Documents in a List* display; option 6 • *Work with Mail* display; option 6
'PRINTOPTS'	Print with options (DOCI0100 format)	• PRTDOC command; Display print options is *YES • *Work with Documents in Folders* display; option 9 • *Work with Documents in a List* display; option 9 • *Work with Mail* display; option 9
'PAGINATE'	Paginate document (DOCI0100 format)	• PAGDOC command • *Work with Documents in Folders* display; option 13

Figure 21-3 (Page 2 of 2). Document Handling Function Requests

Function	Description	Where Called From
'SPELLCHECK'	Check the spelling accuracy or grade level of a document (DOCI0300 format)	• CHKDOC command • *Work with Documents in Folders* display: option 11

DOCI0100 Format

This is the function-specific information for PRINT and PRINTOPTS requests. This format is filled out the same as it would have looked if the OfficeVision/400 program were to process the request. This format merges the parameters specified on the PRTDOC command with the print options record. The *Field* column describes the associated PRTDOC parameter for each format. The *Office Services Concepts and Programmer's Guide* has more information about the meaning of the parameters.

Refer to the "Field Descriptions" on page 21-12 for more details about the fields in this format.

Figure 21-4 (Page 1 of 2). Document Handling Values for Print Function Requests

Offset			
Dec	Hex	Type	Field
0	0	CHAR(2)	Number of copies to print
2	2	CHAR(1)	Output device
3	3	CHAR(10)	Printer device name
13	D	CHAR(10)	Output queue name
23	17	CHAR(10)	Output queue library name
33	21	CHAR(10)	Output file name
43	2B	CHAR(10)	Form type
53	35	CHAR(10)	Printer file name
63	3F	CHAR(10)	Printer file library name
73	49	CHAR(10)	Device file name
83	53	CHAR(10)	Device library name
93	5D	CHAR(10)	Device member name
103	67	CHAR(1)	Delay printing
104	68	CHAR(1)	Draft spacing
105	69	CHAR(1)	Print line numbers
106	6A	CHAR(1)	Resolve instructions
107	6B	CHAR(1)	Large print
108	6C	BINARY(4)	Graphic character set
112	70	BINARY(4)	Code page
116	74	CHAR(1)	Print separator page
117	75	CHAR(1)	Adjust line endings
118	76	CHAR(1)	Adjust page endings

Figure 21-4 (Page 1 of 2). Document Handling Values for Print Function Requests

Offset			
Dec	Hex	Type	Field
119	77	CHAR(1)	Allow widow lines
120	78	CHAR(2)	Additional spaces to left
122	7A	CHAR(1)	Print change symbols
123	7B	CHAR(5)	Change symbols to print
128	80	CHAR(1)	Print quality
129	81	CHAR(1)	Place on job queue
130	82	CHAR(1)	Send completion message
131	83	CHAR(10)	Job description name
141	8D	CHAR(10)	Job description library name
151	97	CHAR(1)	Cancel on error
152	98	CHAR(1)	Print error log
153	99	CHAR(10)	Error log form type
163	A3	CHAR(1)	Clear error log after printing
164	A4	CHAR(1)	Save resolved output
165	A5	CHAR(12)	Resolved output document
177	B1	CHAR(63)	Resolved output folder
240	F0	CHAR(1)	Multiple line report
241	F1	CHAR(1)	Print on both sides
242	F2	CHAR(1)	Automatic page binding
243	F3	CHAR(1)	Automatically shift margins to avoid truncating text
244	F4	CHAR(1)	Renumber system page numbers
245	F5	CHAR(7)	Print from page 1
252	FC	CHAR(8)	Print to page 1
260	104	CHAR(7)	Print from page 2
267	10B	CHAR(8)	Print to page 2
275	113	CHAR(7)	Print from page 3
282	11A	CHAR(8)	Print to page 3
290	122	CHAR(7)	Print from page 4
297	129	CHAR(8)	Print to page 4
305	131	CHAR(7)	Print from page 5
312	138	CHAR(8)	Print to page 5
320	140	CHAR(7)	Print from page 6
327	147	CHAR(8)	Print to page 6
335	14F	CHAR(7)	Print from page 7
342	156	CHAR(8)	Print to page 7
350	15E	CHAR(1)	Document is a label document
351	15F	CHAR(2)	Number of labels
353	161	CHAR(3)	Width of labels
356	164	CHAR(1)	Sheet-feed labels

Figure 21-4 (Page 2 of 2). Document Handling Values for Print Function Requests

Offset			
Dec	**Hex**	**Type**	**Field**
357	165	CHAR(2)	Number of rows per sheet
359	167	CHAR(1)	Merge type
360	168	CHAR(10)	Query name
370	172	CHAR(10)	Query library name
380	17C	CHAR(12)	Data document name
392	188	CHAR(63)	Data folder name
455	1C7	CHAR(10)	Data file name
465	1D1	CHAR(10)	Data file library name
475	1DB	CHAR(10)	Data file member name

DOCI0200 Format

This is the structure that is used by this exit program when the function is for merge or merge with options. Note that the single-record merge is not supported.

The merge situations where the user specified DSPOPT(*VIEW) or DSPOPT(*EDIT) results in the exit program being called twice, once for the merge and then once for the subsequent view or edit. The *Field* column describes the associated MRGDOC parameter for each field. The *Office Services Concepts and Programmer's Guide* has more information about the meaning of the parameters.

The "Field Descriptions" on page 21-12 have more detailed information about the fields for this format.

Figure 21-5. Document Handling Values for Merge Function Requests

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(12)	To document name
12	C	CHAR(63)	To folder name
75	4B	CHAR(1)	Replace existing document
76	4C	CHAR(1)	Place on job queue
77	4D	CHAR(1)	Send completion message
78	4E	CHAR(10)	Job description name
88	58	CHAR(10)	Job description library name
98	62	CHAR(1)	Merge type
99	63	CHAR(10)	Query name
109	6D	CHAR(10)	Query library name
119	77	CHAR(12)	Data document name
131	83	CHAR(63)	Data folder name
194	C2	CHAR(10)	Data file name
204	CC	CHAR(10)	Data file library name
214	D6	CHAR(10)	Data file member name
224	E0	CHAR(1)	Adjustment option

Figure 21-5. Document Handling Values for Merge Function Requests

Offset			
Dec	**Hex**	**Type**	**Field**
225	E1	CHAR(1)	Multiple line report
226	E2	CHAR(1)	Collect footnotes
227	E3	CHAR(313)	Reserved

DOCI0300 Format

The "Field Descriptions" on page 21-12 have more detailed information about the fields for this format.

Figure 21-6. Document Handling Values for Spell Function Requests

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Type of check
1	1	CHAR(7)	Beginning page number
8	8	CHAR(7)	Ending page number
15	F	CHAR(470)	Reserved

DOCI0400 Format

The "Field Descriptions" on page 21-12 explain the values to use for mail function requests.

Figure 21-7. Document Handling for Mail Function Requests

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(12)	Mail reference document name
12	C	CHAR(63)	Mail reference folder name
75	4B	CHAR(1)	Attach mail reference
76	4C	CHAR(1)	Same note
77	4D	CHAR(408)	Reserved

DOCI0500 Format

The "Field Descriptions" on page 21-12 explain the values to use for edit function requests.

Figure 21-8. Document Handling for Edit Function Requests

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Show exit display
1	1	CHAR(484)	Reserved

DOCI0600 Format

The "Field Descriptions" on page 21-12 explain the values to use for create function requests.

Figure 21-9. Document Handling for Create Function Requests

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Display exit display
1	1	CHAR(443)	Display document details display
2	2	CHAR(1)	Edit document
3	3	CHAR(482)	Reserved

Field Descriptions

Additional spaces to left. The number of spaces added to the left margin in your printed document. If the document is not printed on both sides of the paper, this is useful when you want to bind your document. Valid values are 0 through 99.

Adjust line endings. The value Y (yes) adjusts line endings in the printed document. The lines are adjusted according to what is specified on the Line Spacing/Justification Options display. This is useful when you print a document that has data merged into it, has instructions, has display attributes that do not print as spaces, or uses a proportionally spaced font.

The value N (no) means you do not want to adjust the line endings in the printed document.

Adjust page endings. The value Y (yes) adjusts page endings in the printed document. The pages are determined by what is specified for the *First typing line* and *Last typing line* prompts on the Page Layout/Paper Options display.

The value N (no) means you do not want to adjust the page endings in the printed document.

Adjustment option. The values indicate the adjustment option you want to use.

1 None
2 Lines (Adjusts line endings in the printed document.)
3 Pages and lines (Adjusts page endings and lines in the printed document.)

Allow widow lines. The value Y (yes) means the page endings or column endings are determined by the exact number of lines per page as specified on the Page Layout/Paper Options display. Also use Y (yes) for the *Adjust page endings* prompt. When you allow widows, a single line from a paragraph could be separated from the rest of the paragraph.

The value N (no) means you do not want to have page endings or column endings determined by the exact number of lines.

Attach mail reference. Whether to concatenate the original mail item with the forward or reply operation. For MAILREPLY, this is entered by the user on the Reply to Mail display. For MAILFWD, this will always be set to a value of yes. The values are yes or no.

Automatic page binding. You can choose to adjust margins for page binding, which is a way to print pages and adjust margins for the even pages so that they will line up with the odd pages when printing on both sides. The values are yes or no.

Automatically shift margins to avoid truncating text. You can automatically shift the margin so that as much text as possible is printed if the margin exceeds the paper edge. The values are yes or no.

Beginning page number. A number from 1.0 through 9999.99 or *FIRST or *LAST to specify the page on which you want spell checking to start.

Cancel on error. The value Y (yes) means you want the document to stop printing if an error is detected during printing. The error is listed in the error log with an error message stating that the job is canceled.

The value N (no) means you want the document to print even if an error is detected during printing.

Change symbols to print. Different revision symbols result in a document when the revision symbol is changed. If your document contains more than one revision symbol character, and you do not select which revision symbol characters you want to print, all revision symbol characters specified in your document will print.

Clear error log after printing. The value Y (yes) means you want previous errors removed from the existing error log page before new ones are added. This is useful if you want the error log to contain only the errors that occurred during one printing of the document.

The value N (no) means you want your errors added to the end of the existing error log page. This is useful if you want to keep a history of all the times your document printed.

Code page. A particular assignment of hexadecimal identifiers to graphic characters. If you want to specify a code page, type the graphic character-set and code-page combination. A code page is a particular assignment of hexadecimal identifiers to graphic characters. When both the graphic character-set ID and code-page ID are typed, they must be separated by a hyphen.

Collect footnotes. The value Y (yes) means you want the associated footnote text for all Footnote instructions found in the included text to become part of the document created during the merge function. All Footnote instructions are changed to refer to the correct system page.

The value N (no) means you want the associated footnote text for all Footnote instructions found in the included text to remain outside of the document created during the merge function.

Data document name. The name of the document that contains the data to be merged.

The document name must be 1 to 12 characters in length.

Data file library name. The name of the library where the data file is located.

Data file member name. The name of the member that contains data.

Data file name. The name of the file to merge data from.

Data folder name. The name of the folder that contains the document that has the data to be merged.

Delay printing. The value Y (yes) delays printing your labels. The labels are held on the output queue where you can release them to print or delete them if you do not want them to print.

Device file name. The name of the file that contains the information about the device. Use the value 3 for file.

Device library name. A user-defined word that names a library. The value is *LIBL.

Device member name. The value *FILE for the member name, means the member with the same name as the file name will be used. The values are *FILE, *FIRST, and *LAST.

Display document details display. Whether or not you want the Document Details display shown. The values are yes or no.

Display exit display. Whether or not you want the exit display shown. The values are yes or no.

Document is a label document. The value (yes or no) identifies if the document is a label document. A label document contains the labels you are printing.

Draft spacing. The value Y (yes) doubles the spacing value in the *Line spacing* prompt on the Line Spacing/Justification Options display. For example, if the *Line spacing* prompt is 3 (Triple), the doubled spacing value is 6 and five blank lines are printed between each line of text in your document.

Edit document. The values (yes or no) indicate whether or not you want to edit the document.

Ending page number. A number from 1.0 through 9999.99 or *FIRST or *LAST to specify the page on which you want spell checking to stop.

Error log form type. The forms type for the type of paper on which you want the error log to be printed. If you use the value *STD as the forms type, the error log will be printed on the paper specified in the printer file for the printer you selected. A printer file contains information controlling how your document is printed on a particular printer.

Form type. The host system form type for the forms control table (FCT) entry.

Graphic character set. The graphic character-set ID that you want to print your job. A graphic character-set ID is an identifier used to specify a set of graphic characters in a code page. This identifier can be two 4-digit prompts separated by a blank or by a hyphen.

Job description library name. The library name by specifying one of the following:

name The name of the library that is storing your job description.

**LIBL* If you use *LIBL, your library list is searched for the job description.

Job description name. The name of the job description that describes how the job is to be run.

Large print. The value Y (yes) prints your document with large print.

The value N (no) indicates that you do not want to print your document with large print.

Mail reference document name. The document name where a mail item is referred.

Mail reference folder name. The folder name where the referenced document is filed.

Merge type. The value identifies the way the documents are merged. The values are:

1 Query
2 Document
3 File and Blank

Multiple line report. The value Y (yes) means you want to create a multiple line report. A multiple line report merges Data Field instructions to create a report where each record of data produces several lines of output.

The value N (no) means you do not want to create a multiple line report.

Number of copies to print. The quantity of copies you want printed. The valid values are from 1 through 99.

Number of labels across page. The value that identifies the amount or quantity of labels on a page. The valid values are from 1 through 99.

Number of rows per sheet. If you selected Y (yes) for the *Sheet feed labels* prompt, this value from 1 through 99, is the number of rows of labels that you want printed on a page.

Output device. A device in a system by which data can be received from the system. The values are:

1 Printer
2 Display
3 File

Output file name. This is the name of the spooled file that is created. The values are *DOC and *FILE.

Output queue library name. The name of the library being used as the current library during the processing of this command. The value is *LIBL.

Output queue name. A list of output files to be printed or displayed. The output queue is used for spooled files. The values are *DEV, *FILE, and *WRKSTN.

Place on job queue. A document can be merged on the job queue.

Use the value Y (yes) if you want the document merged on the job queue. You can then continue working on your display while the document is being merged on the job queue.

Use the value N (no) if you want the job to merge interactively.

Print change symbols. The value Y (yes) prints revision symbols in the left margin of your document. Revision or change symbols are used to indicate lines that have been revised since the last time the document was changed. The revision symbol character is specified on the Change Editing Options display.

Use N (no) if you do not want to print the revision symbols in the left margin of your document.

Print error log. The value Y (yes) prints the error log page at the end of your document. The log contains any errors that were found while the document was being prepared for printing or errors found when a document was sent from another system. The log can also contain informational messages. If your document does not contain print errors, no log will be printed.

Print from pages 1 through 7. A number from 0.01 through 9999.99 to specify the page on which you want printing to start. Other valid values are *FIRST, *LAST, or *STRPAGE. If you use *FIRST, printing is started on the first page of the document. If you use *LAST, printing is started on the last page of the document. If you use *STRPAGE, the to and from page values are the same and only one page is printed. The *From page* and *To page* prompts tell the printer to print specific pages from your document. The page values specified are the pages from the printed document.

Print line numbers. The value Y (yes) prints line numbers in your document. The line numbers begin with 1 on the first page of your document. Line numbers are not printed in headers or footers.

The value N (no) means you do not want to print line numbers in your document.

Print on both sides. The value indicates if you want your document printed on one side or both sides of a page. The values are:

1 No
2 Yes
3 Tumble

Print quality. The value indicates the quality of printing you want to use. The values are:

1 Letter

Letter-quality type is better print but causes wear on the printer ribbon (if your printer uses a ribbon) because the ribbon is struck harder.

2 Text

Text-quality type is not as good as letter-quality type, but is better than draft-quality type.

3 Draft

Draft-quality type saves wear on the printer ribbon (if your printer uses a ribbon) because the ribbon is not struck as hard as it would be if you were using letter-quality type or text-quality type.

Print separator page. A value that determines if sheets will separate the jobs in the printer. The value Y (yes) prints a separator page that includes such things as the document name, folder, document description, subject, reference, and authors. This is useful for separating your document from other documents.

Use N (no) if you do not want to print a separator page.

Print to pages 1 through 7. A number from 0.01 to 9999.99 is the page on which you want printing to stop. Other valid values are *FIRST and *LAST. If you use *FIRST, printing is ended on the first page of the document. If you use *LAST, printing is ended on the last page of the document. The *From page* and *To page* prompts tell the printer to print specific pages from your document. The page values specified are the pages from the printed document.

Printer device name. A device that writes output data from a system on paper or other media.

Printer file library name. The library name of the printer file. Possible values are:

name The name of the library in which the printer file is stored.
**LIBL* If you specify *LIBL, your library list is searched first for the printer file.

Printer file name. The name of the file where you want the printed labels to be received and stored. A file is a group of records that are related and treated as a unit. If the file does not exist, a new file will be created for you. If the file already exists, the document will be added to the end of the file.

Query library name. The name of the library that contains the query.

Query name. The name of the query that contains data to be merged.

Renumber system page numbers. The value Y (yes) renumbers the system page numbers.

The value N (no) keeps the same system page numbers.

Replace existing document. Whether the document replaces the existing document (yes) or is added to the existing documents (no).

Reserved. An ignored field.

Resolve instructions. The value Y (yes) processes the instructions that you have placed in your document.

Resolved output document. The saved document is a resolved document (saved in printed form), that is, page endings are adjusted, line endings are adjusted, headers and footers are put in, and instructions are processed (optional). If you wish to print this document again, it takes less time to print because this document is already in printed form. (A resolved document is a document with the text instructions processed.) This document can be from the text editor or another system.

Resolved output folder. The name of the folder that will contain the document you specified in the *Document name* prompt. If the folder does not exist, a message is shown before it is created.

Same note. Whether the document handling program has previously been called to process this forward or reply operation. This indication provides a means for the document handling program to distinguish between the initial and subsequent calls to process this document. This can be used to avoid attaching the mail reference multiple times. The valid values are yes or no.

Save resolved output. The value Y (yes) means the document you are printing is saved in the document specified in the document prompt.

The value N (no) means you do not want the document you are printing saved in another document.

Send completion message. The value Y (yes) means you are putting your print job on the job queue and you want a message sent to your display station when the job has completed.

The value N (no) means you are putting your print job on the job queue and you do not want a message sent to your display station when the job has completed.

Sheet-feed labels. A value that defines that a device is attached to a printer to automatically feed out sheets of labels. The value Y (yes) means you are sheet-feed printing and want more than one row of labels on a page. If you are using sheet-feed paper, there is no other way to print more than one row of labels on a page.

Show exit display. Whether or not you want the exit display shown. The valid values are yes or no.

To document name. The name of the document to be merged with.

To folder name. The name of the folder that will contain the document being created.

Type of check. The value indicates if you want to check the spelling or the grade level.

0 Spell
1 Grade

Width of labels. The width of a label is the number of characters from the left edge of the first label to the left edge of the next label, including the blank spaces between the labels. If the width you specify is larger than the margins for your document, the margins are used as the width. Valid values are 2 through 198.

Note: For the printer device field the special values *SYSVAL, *USRPRF, and *WRKSTN are replaced with the appropriate printer name, therefore the exit program does not use the special values.

Error Messages

OFC1680	Message returned from document handling program.
OFC1690	OfficeVision/400 processing requested by program &1 in library &2.
OFC1691	Error while calling program &1 in library &2.

Part 10. Operational Assistant APIs

Chapter 22. Operational Assistant APIs

Most functions on the AS/400 Operational Assistant* menu can be accessed individually by calling APIs found in the QSYS library. These Operational Assistant APIs allow you to incorporate Operational Assistant functions into your application menus.

Your assistance level setting affects the type of display you see when these APIs are called.

The APIs in this chapter are presented in alphabetical order.

The Operational Assistant APIs are:

Operational Assistant Attention-Key-Handling (group jobs) (QEZMAIN) creates a group job to display the AS/400 Operational Assistant menu.
Operational Assistant Attention-Key-Handling (nongroup jobs) (QEZAST) uses the GO ASSIST command to display the AS/400 Operational Assistant menu.
Save Information (QEZSAVIN) displays the Save Information to Help Resolve a Problem display.
Send Message (QEZSNDMG) displays the Operational Assistant Send a Message display.
Work with Jobs (QEZBCHJB) displays either the Work with Jobs display or the Work with User Jobs display.
Work with Messages (QEZMSG) displays either the Work with Messages display or the Display Messages display.
Work with Printer Output (QEZOUTPT) displays either the Work with Printer Output display or the Work with All Spooled Files display.

Operational Assistant Attention-Key-Handling (group jobs) (QEZMAIN) API

The Operational Assistant Attention-Key-Handling (QEZMAIN) API creates a group job to display the AS/400 Operational Assistant menu (ASSIST). This avoids the Attention-key-handling program running in the same job as an application which leaves the keyboard unlocked between input and output operations. This API can only be used in an interactive job.

Notes:

1. The various job attributes (such as library list or the System/36 environment) may differ between the user's job and the job in which the ASSIST menu runs. To avoid this situation, you might want to use the QEZAST API.
2. This program is only intended to be used as an Attention-handling program for jobs that do not transfer to other group jobs.

There are no parameters for this API.

Error Messages

CPF1E15 E Problem occurred while calling Operational Assistant.

Operational Assistant Attention-Key-Handling (nongroup jobs) (QEZAST) API

The Operational Assistant Attention-Key-Handling (QEZAST) API uses the GO ASSIST command to display the AS/400 Operational Assistant menu (ASSIST). If you want your users to access the Operational Assistant menu by selecting an option from your application menus, you can add the control language (CL) statement ***call qezast*** to your application. This API can only be used in an interactive job.

This API can be used in place of the Operational Assistant Attention-Key-Handling (QEZMAIN) API when you do not want the Attention key to bring up the ASSIST menu in a group job.

There are no parameters for this API.

Error Messages

CPF6ACD E Menu &1 in &2 is wrong version for system.
CPF6AC6 E System started the display menu program again.

Save Information (QEZSAVIN) API

The Save Information (QEZSAVIN) API displays the Save Information to Help Resolve a Problem display (option 10 on the Operational Assistant Documentation and Problem Handling menu). On that display, users can type a short description of the problem that they are experiencing with the system or with an application. A problem ID is assigned so that a problem analysis person can later look at this information using the Work with Problem (WRKPRB) command. The following information is collected:

- The entries in the QHST history log for the previous hour.
- Printer output from the following commands is placed on the QEZDEBUG output queue:
  - Work with Active Jobs (WRKACTJOB)
  - Display Messages (DSPMSG—For the work station and user)
  - Display System Operator Messages (DSPMSG QSYSOPR)
  - Display Job Log (DSPJOBLOG—For each group job at your display station)
  - Display Job (DSPJOB—For each group job at your display station)
  - Display PTF (DSPPTF—*ALL to give the PTF level of your system)

- Any service dumps (QPSRVDMP), program dumps (QPGMDMP), and job logs (QPJOBLOG) for this user.

There are no parameters for this API. This API can only be used in an interactive job.

Error Messages

CPF1E99 E Unexpected error occurred.
CPF9871 E Error occurred while processing.

Send Message (QEZSNDMG) API

Parameters

Optional Parameter Group:

1	Message type	Input	Char(10)
2	Delivery mode	Input	Char(10)
3	Message text	Input	Char(*)
4	Length of message text	Input	Binary(4)
5	List of user profile names	Input	Array of Char(10)
6	Number of user profile names	Input	Binary(4)
7	Message sent indicator	Output	Binary(4)
8	Function requested	Output	Binary(4)
9	Error code	I/O	Char(*)

The Send Message (QEZSNDMG) API displays the Operational Assistant Send a Message display (Figure 22-1). Parameters can be specified, providing the ability to display initial values (defaults) to the user. Parameters determine the type of message that is sent (informational or inquiry), the delivery mode of the message (break or normal), the message text, and the users who will receive the message.

This API combines the functions of the Send Message (SNDMSG) and Send Break Message (SNDBRKMSG) commands. In addition, it provides the ability to send inquiry messages to more than one user, to send break messages to user profiles, and to send break and inquiry messages to all active users.

This API can be called with or without parameters. If parameters are specified, all parameters are required. This API can only be used in an interactive job.

Refer to Chapter 19, "Network Management APIs" on page 19-1 for information on alert messages. Refer to Chapter 18, "Message Handling APIs" on page 18-1 for information on the message handling APIs.

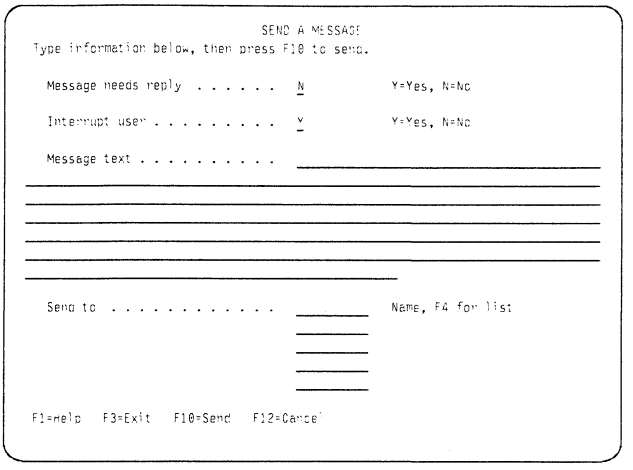


Figure 22-1. Send a Message Display

Optional Parameter Group

Message type
INPUT; CHAR(10)
The type of message to send. The value you specify determines the default for the *Message needs reply* prompt on the Send a Message display (N for *INFO and Y for *INQ). You must specify one of these values:

***INFO** Informational. The message does not need a reply.

***INQ** Inquiry. The message needs a reply. The reply is placed on the message queue specified in the user profile of the sender. The user profile name of the person sending the reply is added to the message text, allowing the person receiving the reply to determine which user it is from.

Delivery mode
INPUT; CHAR(10)
The delivery mode of the message. The value you specify determines the default for the *Interrupt user* prompt on the Send a Message display (Y for *BREAK and N for *NORMAL). If the user is not authorized to send a break message, *NORMAL is used regardless of the value you specify. You must specify one of these values:

***BREAK** Break message. If the user is signed on, the message goes to the work station message queues that the user is signed on to and temporarily interrupts the work that the user is doing. If the user is not signed on, the message goes to the user profile message queue and the sender is notified.

***NORMAL** The message goes to the user profile message queue. If the message queue is in notify mode for that user, the message waiting light is turned on. If the message queue is in break mode, the message temporarily interrupts the work that the

receiver is doing. If the message queue is in hold mode, the receiver is not notified.

Message text
INPUT; CHAR(*)
The complete text of the message. The text you specify is displayed as the default on the Send a Message display.

Length of message text
INPUT; BINARY(4)
The length of the message text, in bytes. Valid values are 0 through 494.

List of user profile names
INPUT; ARRAY OF CHAR(10)
A list of 0 through 299 user profile names to wnich the message is being sent. The list you specify is displayed as the default on the Send a Message display.

The message is sent to the user profile message queue. To specify other message queues, use one of the following special values:

***ALL** The message queues of all users. When you use this value, it must be the only item in the list.

***ALLACT** The message queues of all active users. This value can be used in combination with specific user profile names and with *SYSOPR.

***SYSOPR** The system operator's message queue, QSYSOPR. This value can be used in combination with specific user profile names and with *ALLACT.

Note: *JOBCTL special authority is required to use the *ALL or *ALLACT value in this parameter.

Number of user profile names
INPUT; BINARY(4)
The number of user profile names specified. Valid values are 0 through 299. When you use the special value *ALL for the list of user profile names parameter, specify 1.

Message sent indicator
OUTPUT; BINARY(4)
Whether the user pressed F10 (Send message) to send one or more messages from the Send a Message display.

One of the following values is returned:

0 No messages were sent.
1 One or more messages were sent.

Function requested
OUTPUT; BINARY(4)
The function that the user requested when exiting the Send a Message display.

One of the following values is returned:

-4 User pressed the Exit key (F3).

-8 User pressed the Cancel key (F12).

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1EBA E Parameters passed on CALL do not match those required.
CPF1EB2 E Delivery mode must be *BREAK or *NORMAL.
CPF1EB3 E Value for length of message text must be 0 through 494.
CPF1EB4 E Number of user profile names not valid.
CPF1EB5 E Value for number of user profile names must be 0 through 299.
CPF1EB6 E Program &1 cannot be run as a batch job.
CPF1EB7 E *ALL not allowed with other user profile names.
CPF1EB9 E Value &1 not valid for list of user profile names.
CPF24B3 E Message type &1 not valid.
CPF3CF1 E Error code parameter not valid.

Work with Jobs (QEZBCHJB) API

The Work with Jobs (QEZBCHJB) API uses the Work with User Jobs (WRKUSRJOB JOBTYPE (*BATCH)) command to display either of the following:

- For the basic assistance level, the Work with Jobs display
- For the intermediate assistance level, the Work with User Jobs display

There are no parameters for this API.

Error Messages

Messages are the same as the WRKUSRJOB command messages. For additional information, refer to the *Programming Reference Summary*.

Work with Messages (QEZMSG) API

The Work with Messages (QEZMSG) API uses the Display Messages (DSPMSG) command to display either of the following:

- For the basic assistance level, the Work with Messages display
- For the intermediate assistance level, the Display Messages display

There are no parameters for this API. When the API is called, the default parameter values are used for the command.

Error Messages

Messages are the same as the WRKMSG command messages. For additional information, refer to the *Programming Reference Summary*.

Work with Printer Output (QEZOUTPT) API

The Work with Printer Output (QEZOUTPT) API uses the Work with Spooled Files (WRKSPLF) command to display either of the following:

- For the basic assistance level, the Work with Printer Output display
- For the intermediate assistance level, the Work with All Spooled Files display

There are no parameters for this API. When the API is called, the default parameter values are used for the command.

Error Messages

Messages are the same as the WRKMSG command messages. For additional information, refer to the *Programming Reference Summary*.

Chapter 23. Operational Assistant Exit Programs

This chapter discusses how to tailor some of the Operational Assistant functions to your needs by using exit programs.

The Operational Assistant exit programs consist of:

Tailoring automatic cleanup (QEZUSRCLNP) for running your own cleanup programs (IBM supplied).
Tailoring Operational Assistant backup for running your own backup functions (user supplied).
Tailoring power off (QEZPWROFFP) for changing how you want the system automatically powered on and off (IBM supplied).

Exit Program for Tailoring Automatic Cleanup

You may want to develop your own programs to regularly clean up some of the objects that are not handled by the Operational Assistant automatic cleanup functions.

You can incorporate your own programs into the IBM-supplied automatic cleanup function by using the QEZUSRCLNP program. Then, whenever the system runs automatic cleanup, it also runs your own cleanup programs.

To make a copy of the QEZUSRCLNP program:

1. Type `RTVCLSRC` (the Retrieve CL Source command) on any command line and press F4 (Prompt). Type the following values for the prompts:

Program	`QEZUSRCLNP`
Library	`QSYS`
Source file	`Name of source file`
Library	`Name of library`

   Press the Enter key.
2. Insert statements that run your own cleanup programs into your copy of QEZUSRCLNP.
3. Compile your copy of the QEZUSRCLNP program and store it in a library that appears before the QSYS library in the system part of the library list as specified in system value QSYSLIBL.

Whenever the system runs the automatic cleanup function, your version of QEZUSRCLP is also run.

Exit Program for Tailoring Operational Assistant Backup

Parameters

Required Parameter Group:

1	Calling product	Input	Char(10)
2	Exit indicator	Input	Char(10)
3	Options	Input	Char(10)
4	Device	Input	Char(40)
5	Tape set	Input	Char(4)
6	Return code	Input	Char(7)

You can tailor the Operational Assistant automatic backup by specifying an exit program on the Change Backup (CHGBCKUP) command. If an exit program is specified, that program is called both before the Operational Assistant backup is run and after backup is run. You specify within the exit program when you want your functions to run.

If this program ends abnormally or sends an escape message to its caller when running before the backup, the backup will not continue.

When running backup, you could write your own exit program to back up some additional objects that are not included on the folder or library backup list.

Another example of using an exit program could be when you want to clean up some items before the system does its backup. This would save time and resources by not backing up objects you want deleted.

Required Parameter Group

Parameters your exit program should be able to handle are:

Calling product
INPUT; CHAR(10)
The name of the product calling the exit program. This parameter is supplied so that the exit program can tell whether it is called from the AS/400 Run Backup (RUNBCKUP) command or from another application. When called from the RUNBCKUP command, the value is QEZBACKUP.

Exit indicator
INPUT; Char(10)
Whether this program is called before or after the backup is done.

The possible values are:

***BEFORE** This call is before the backup has started.
***AFTER** This call is after the backup has run.

Options
INPUT; Char(10)
Indicates that the specified backup options are used.

The possible values are:

***DAILY** The daily backup options are used.
***WEEKLY** The weekly backup options are used.

***MONTHLY** The monthly backup options are used.

Device
Input; Char(40)
The name of up to four devices to be used for the backup. Each device is left-justified on a 10-byte boundary.

Tape set
Input; Char(4)
The name of the tape set to be used for the backup. Operational Assistant backup combines the tape set name (1 to 4 characters) with volume numbers from 01 to 99 to generate the volume IDs of the tape volumes to be used by the backup.

Return code
Input; Char(7)
The message ID of the message returned by backup. This is blank before the backup.

Error Messages

CPC1E62 C Backup successfully completed.
CPF1E68 E Backup incomplete.
CPF1EE7 E Unexpected error occurred during backup.
CPF1E99 E Unexpected error occurred.

Exit Program for Tailoring Power Off

The Power-Off exit program (QEZPWROFFP) is shipped with the system and stored in the QSYS library. This exit program powers off the system according to the power on and off schedule by running the Power Down System command PWRDWNSYS OPTION(*IMMED). (Use the Change Power Schedule Entry (CHGPWRSCDE) command or the Change Power Schedule (CHGPWRSCD) command to set the power on and off schedule.)

You can tailor this program to change how you want the system powered off. For example, you could change the program so that the system would not power off immediately.

To change this program:

1. Type RTVCLSRC (the Retrieve CL Source command) on any command line and press F4 (Prompt). Type the following values for the prompts:

Program	QEZPWROFFP
Library	QSYS
Source file	Name of source file
Library	Name of library

Press the Enter key.

2. Change the source to tailor the program. For example, you may want to specify RESTART(*YES) for the PWRDWNSYS command to cause the system to power on (IPL) when the power down is complete.
3. When you are finished changing the program, use the Create CL Program (CRTCLPGM) command and fill in the fields as you did when you retrieved the program. The program you create must be named QEZPWROFFP.
4. To use the new program, put the new program in a library before the QSYS library in the system part of the library list as specified in system value QSYSLIBL.

Notes:

1. Remember that this program controls the scheduled powering off of the system. If you remove the Power Down System (PWRDWNSYS) command, the system does not automatically power down on schedule. If you change the OPTION parameter on the PWRDWNSYS command to *CNTRLD, the system may wait the amount of time specified by the DELAY parameter before it powers off. If the DELAY parameter is changed to *NOLIMIT, the system may never power off.
2. If your next scheduled power-on time is fairly close to the scheduled power-off time, and you have long running commands or calls to programs ahead of the PWRDWNSYS command, your system may not have powered off by the time it is scheduled to power on again. Before the system is powered down, check the time of the next scheduled power up using the Retrieve Power Schedule Entry (RTVPWRSCDE) command. You should also use the Change Power Schedule Entry (CHGPWRSCDE) command to reset the system value. If it is within 1/2 hour, do not power off the system; otherwise, your system may not power on again, and you will need to do a manual IPL.
3. This program will not get called when the current time is less than 30 minutes before the next scheduled power-on time.

Part 11. Program and CL Command APIs

Chapter 24. Program and CL Command APIs

This chapter describes the APIs that you can use to create programs, to retrieve program information, and to retrieve command information. The APIs are:

Create Program (QPRCRTPG) converts the symbolic representation of a machine interface (MI) program into a program object.

Retrieve Command Information (QCDRCMDI) retrieves information from a command definition object and places it into a single variable in the calling program. For information about this API, see "Retrieve Command Information (QCDRCMDI) API" on page 24-18.

Retrieve Program Information (QCLRPGMI) retrieves program information and places it into a single variable in the calling program. For information about this API, see "Retrieve Program Information (QCLRPGMI) API" on page 24-22.

Also included in this chapter are programming tips for using the QPRCRTPG API, which include:

- MI program syntax, including labels and instruction, declare, and directive statements
- MI program coding techniques, such as how to use declare statements, space objects, and constants

Before using this information, you should have some MI programming experience and understand the concepts in the *MI Functional Reference*.

Create Program (QPRCRTPG) API

Parameters

Required Parameter Group:

1	Intermediate representation of the program	Input	Char(*)
2	Length of intermediate representation of program	Input	Binary(4)
3	Qualified program name	Input	Char(20)
4	Program text	Input	Char(50)
5	Qualified source file name	Input	Char(20)
6	Source file member information	Input	Char(10)
7	Source file last changed date and time information	Input	Char(13)
8	Qualified printer file name	Input	Char(20)
9	Starting page number	Input	Binary(4)
10	Public authority	Input	Char(10)
11	Option template	Input	Char(*)
12	Number of option template entries	Input	Binary(4)

Optional Parameter:

13	Error code	I/O	Char(*)

The Create Program (QPRCRTPG) API converts the symbolic representation of a machine interface (MI) program into a program object. This symbolic representation is known as the intermediate representation of a program.

The QPRCRTPG API creates a program object that resides in the *USER domain and runs in the *USER state. If you want the program object to be temporary, you must do one of the following:

- Delete the object when you no longer need it.
- Create the object in the QTEMP library, and let the system delete the object automatically when the job ends.

You can specify program objects created with the QPRCRTPG API in CL commands that process objects of type *PGM. For example, you can:

- Save and restore program objects using the Save Object (SAVOBJ) and Restore Object (RSTOBJ) commands.
- Delete program objects using the Delete Program (DLTPGM) command.
- Run program objects using the Call (CALL) command.
- Rename program objects using the Rename Object (RNMOBJ) command.
- Move program objects to a different library using the Move Object (MOVOBJ) command.

Authorities and Locks

Program Authority
*ALL. Required only if the program already exists and the option value *REPLACE is specified.

Program Library Authority
*CHANGE

Printer File Authority
*USE

Printer File Library Authority
*USE

Required Parameter Group

Intermediate representation of the program
INPUT; CHAR(*)
A string containing the intermediate representation of the program to be processed by the QPRCRTPG API. See "Program Syntax" on page 24-6.

Length of intermediate representation of program
INPUT; BINARY(4)
The size, in bytes, of the intermediate representation of the program.

Qualified program name
INPUT; CHAR(20)
The name and library of the program to be created or replaced. The first 10 characters contain the program name, and the second 10 characters contain the name of the library where the program is located. The special value *CURLIB may be used for the library name.

Program text
INPUT; CHAR(50)
Text that briefly describes the program.

Qualified source file name
INPUT; CHAR(20)
The name and library containing the source program. The first 10 characters contain the source file name, and the second 10 characters contain the name of the library where the file is located. This places the value in the program object's service description. The special value *NONE may be used for the source file name. If you specify *NONE, no source file information is placed in the program object's service description. A special value, such as *LIBL, is not valid for the source file library.

Source file member information
INPUT; CHAR(10)
The file member containing the source program. This places the value in the program object's service description.

This value must be blanks if you specify *NONE as the **source file name**.

Source file last changed date and time information
INPUT; CHAR(13)
The date and time the member of the source file was last updated. The format of this field is in the CYYMMDDHHMMSS format, where:

C	Century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
YY	Year
MM	Month
DD	Day
HH	Hour
MM	Minute
SS	Second

This places the value in the program object's service description.

This value must be blank if you specify *NONE for the source file name parameter.

Qualified printer file name
INPUT; CHAR(20)
The name and library containing the printer file used to generate listings. The first 10 characters contain the printer file name, and the second 10 characters contain the name of the library where the file is located. The only special values supported for the library name are *LIBL and *CURLIB.

This value is ignored if you specify *NOLIST for the generate listing option (see "Values for the Option Template Parameter").

Starting page number
INPUT; BINARY(4)
The first page number to be used on listings. This value should be between 1 and 9999; otherwise, the API uses 1.

This value is ignored you specify *NOLIST for the generate listing option (see "Values for the Option Template Parameter").

Public authority
INPUT; CHAR(10)
The authority you give the users who do not have specific private authorities to the object, and where the user's group has no specific authority to the object. The values allowed are:

*CHANGE
*ALL
*USE
*EXCLUDE
The name of an authorization list

Option template
INPUT; CHAR(*)
This is an array of options. You can specify between 0 and 16 values. Each entry contains a CHAR(11) value as described in "Values for the Option Template Parameter."

Number of option template entries
INPUT; BINARY(4)
The number of option template entries. The value must be between 0 and 16.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Values for the Option Template Parameter

When you are using the QPRCRTPG API, you can specify a value in the option template. Only one value per option should be specified. If you specify more than one, the system only uses the first one. If you specify no value for a given option, the system uses the default value (underlined).

Create program object
Creates a program object.

The values allowed are:

<u>*GEN</u>	Generates a program and places the program in the appropriate library.
***NOGEN**	No program is generated. The syntax of the intermediate representation of the program is checked, and if the generate listing option is *LIST, a listing is produced.

Replace program
Replaces the existing program if a program by the same name already exists in the specified library.

The values allowed are:

***NOREPLACE** Does not replace an existing program by the same name in the specified library.

***REPLACE** Replaces the existing program by moving it to the QRPLOBJ library.

Generate listing

Generates an output listing.

The values allowed are:

***NOLIST** Does not generate a listing.

***LIST** Generates a listing. You must specify the following parameters:

- Printer file name and library
- Starting page number

Create cross-reference listing

Whether the listing is to contain a cross-reference list of variable and data item references.

The values allowed are:

***NOXREF** Does not create cross-reference listing.

***XREF** Creates a cross-reference listing of references to variables, labels, or both.

Create summary listing

Whether the listing is to contain a list of program attributes.

The values allowed are:

***NOATR** Does not create a summary listing section.

***ATR** Creates a summary listing section.

User profile

The values allowed are:

***USER** The user profile of the user running the program is used as a source of authority when this program runs.

***ADOPT** When the program runs, the object authority of both the program's owner and user are used.

***OWNER** The system uses the user profile of the owner of the program as a source of authority when this program runs. Programs called by this program adopt this authority.

Use adopted authority

Whether the system uses the program-adopted authority from the calling programs as a source of authority when this program is running.

The values allowed are:

***ADPAUT** The system uses program-adopted authority.

***NOADPAUT** The system does not use program-adopted authority.

Constrain arrays

The values allowed are:

***SUBSCR** Constrains arrays. This requests additional run-time checks to ensure that references to array elements are not outside the bounds of the declare statement. This option causes the resulting program to run slower.

***NOSUBSCR** Does not constrain arrays. The results of references to array elements outside the bounds of the declare statement are not defined.

***UNCON** Allows fully unconstrained arrays. This ensures that references to array elements outside the bounds of the declare statement act as if the array element actually exists.

Note: This program attribute may be changed at run-time using the Override Program Attributes (OVRPGATR) MI instruction.

Constrain strings

The values allowed are:

***SUBSTR** Constrains strings. This requests additional run-time checks to ensure that references to character strings are not outside the bounds of the declare statement. This option causes the resulting program to run slower.

***NOSUBSTR** Does not constrain strings. The results of substring references outside the bounds of the declare statement are not defined.

Note: You can change this program attribute at run-time using the Override Program Attributes (OVRPGATR) MI instruction.

Initialize static storage

Static storage is allocated the first time a program is called. It remains allocated until explicitly deallocated.

The values allowed are:

***CLRPSSA** Initializes static storage. This code clears the program static storage area (PSSA) on entry using the Call External (CALLX) MI instruction.

***NOCLRPSSA** Does not initialize the PSSA.

Initialize automatic storage

Automatic storage is allocated each time a program runs and automatically deallocated when no longer needed.

The values allowed are:

***CLRPASA** Initializes automatic storage. This code clears the program automatic storage area (PASA) on entry using the Call External (CALLX) MI instruction.

***NOCLRPASA** Does not initialize the PASA.

Ignore decimal data errors

Whether errors found in decimal data result in exceptions.

The values allowed are:

***NOIGNDEC** Does not ignore decimal data errors.

When you specify *NOIGNDEC, decimal values used in numeric operations are checked for valid decimal digits and sign codes. If the system finds an error, it signals an exception.

***IGNDEC** Ignores data decimal errors.

When you specify *IGNDEC, decimal values used in numeric operations ensure they contain valid decimal digit and sign codes. However, the system treats digits that are not valid as zeros and signs that are not valid as positive signs. There is no exception signaled.

This option applies to only a subset of the numeric operations you specify.

Note: In all cases, the system signals decimal data errors if you use data pointers to address any of the instruction's operands.

The following list contains the MI instructions this option affects:

MI Instruction	Packed Source Operands Supported	Zoned Source Operands Supported	Notes
ADDN		X	
CMPNV		X	
CVTCN	X	X	You must specify operand 3 (the numeric view to be used for operand 2) as a constant and no data-pointer-defined operands.
CVTDFFP		X	
CVTNC	X	X	You must specify operand 3 (the numeric view to be used for operand 1) as a constant and no data-pointer-defined operands.
CPYNV	X	X	You must specify no data-pointer-defined operands.
DIV		X	
DIVREM		X	
EDIT		X	You must specify no data-pointer-defined operands.
EXTRMAG		X	
MULT		X	
NEG		X	
REM		X	
SCALE		X	
SUBN		X	

When you specify *IGNDEC, the system may still signal the decimal data exception. That is, other MI instructions and instruction combinations not listed above may signal the decimal data exception when the system finds decimal data that is bad.

Ignore binary data size errors

The values allowed are:

***NOIGNBIN** The system handles binary data size errors normally. When a binary size error occurs, an exception is signaled and the receiver contains the left-truncated result.

***IGNBIN** The system ignores binary data size errors. This is used when an overflow or underflow occurs on a computation and when a control MI instruction has a receiver that is a binary field. The receiver contains the left-truncated result.

Support coincident operands

The system overlaps coincident operands between the source and receiver operands in one or more program instructions. **Coincident operands** are operands that overlap physically, in storage.

The values allowed are:

***NOOVERLAP** Does not support coincident operands. If you specify *NOOVERLAP, coincident operand overlap will not occur while running the instruction. Therefore, the system can use the receiver on an instruction as a work area during operations performed to produce the final result. Using the receiver as a work area does not use as much processor resource as would be required to move the final result from an internal work area to the receiver.

***OVERLAP** Supports coincident operands. If you specify *OVERLAP, the system may overlap coincident operands. Therefore, the system cannot use the receiver on an instruction as a work area during operations that produce the final result. This can require more processor resource for running the instruction but it ensures valid results if an overlap occurs.

The following is a list of instructions this option affects:

- Add logical character (ADDLC)
- Add numeric (ADDN)
- And (AND)
- Compute math function using one input value (CMF1)
- Concatenate (CAT)
- Convert character to numeric (CVTCN)
- Convert decimal form to floating-point (CVTDFFP)
- Convert external form to numeric value (CVTEFN)

- Convert floating-point to decimal form (CVTFPDF)
- Convert numeric to character (CVTNC)
- Copy bytes left adjusted with pad (CPYBLAP)
- Copy bytes right adjusted with pad (CPYBRAP)
- Divide (DIV)
- Divide with remainder (DIVREM)
- Exclusive or (XOR)
- Multiply (MULT)
- Or (OR)
- Remainder (REM)
- Scale (SCALE)
- Subtract logical character (SUBLC)
- Subtract numeric (SUBN)
- Trim length (TRIML)

Allow duplicate declares
The values allowed are:

***NODUP** This does not allow a program object to be declared more than once. This requests that duplicate declare (DCL) statements be diagnosed as errors.

***DUP** This allows a program object to be declared more than once. This requests that program objects declared more than once be pooled and not be diagnosed as errors.

Optimize
The values allowed are:

***OPT** This optimizes the program. In most instances, this produces the smallest and best running program. Occasionally, the source program may signal a MCH2802 escape message during processing. If this occurs, you should not optimize the program.

***NOOPT** This does not optimize the program. This requests the normal level code optimization when you create the program.

Error Messages

CPD0078 E Value &3 for parameter &2 not a valid name.
CPF2143 E Cannot allocate object &1 in &2 type *&3.
CPF2144 E Not authorized to &1 in &2 type *&3.
CPF2146 E Owner of new program and existing program not the same.
CPF2283 E Authorization list &1 does not exist.
CPF3C35 E Value &3 for parameter &2 not a valid name.
CPF3C5A E Number of option template entries is not valid.
CPF3C5B E Option template entry is not valid.
CPF3C5C E Source file name and library is not valid.
CPF3C5D E Source file member is not valid.
CPF3C5F E Internal Representation of Program (IRP) string length parameter is not valid.
CPF3C50 E Program &1 not created.
CPD0078 D Value &3 for parameter &2 not a valid name.
CPD3C50 D Internal Representation of Program (IRP) string length parameter is not valid.
CPD3C52 D Number of option template entries is not valid.
CPD3C53 D Option template entry is not valid.
CPD3C54 D Source file name and library is not valid.
CPD3C55 D Source file member is not valid.
CPD3C56 D Source file last changed date and time is not valid.
CPF3C56 E Source file last changed date and time is not valid.
CPF3C60 E Program name and library is not valid.
CPF3C61 E Authority is not valid.
CPF3C62 E Source file library specified.
CPF3C63 E Source file member specified.
CPF3C64 E Source file last changed date and time specified.
CPF6301 E Intermediate representation of program (IRP) contains &1 errors. Probable compiler error.
CPF6303 E Message &1, &2 received while running create program command.
CPF6304 E Library &1 not found.
CPF6306 E Program &1 in library &2 already exists.
CPF6307 E Program template value at offset &1, bit &2, length &3 not valid.
CPF6308 E Not authorized to create program.
CPF6309 E Not authorized to library &1.
CPF6455 E Member &2 file &1 in library &3 not found.
CPF6457 E Cannot allocate library &1 for program insertion.
CPF6551 E Work space &2 cannot be extended. Probable compiler error.
CPF6552 E Space &2 type &3 subtype &4 not PRM workspace.
CPF6553 E PRM permanent table resolution failed. Probable compiler error.
CPF6554 E Type of IST object &4 at offset &3 not valid. Probable PRM error.
CPF6555 E Addressability field type not valid for IST number &4 at offset &3. Probable PRM error.
CPF6557 E Error condition for IST &4 at &3 of IST space not valid. Probable PRM error.
CPF6560 E Operation code &5 in MI instruction &3 at offset &6 not found in QPRODT.
CPF6561 E Operand &4 in &3 at offset &5 in program template not valid.
CPF6563 E Program was too large to be created.
CPF6564 E Machine storage limit violation.
CPF6565 E User profile storage limit exceeded.

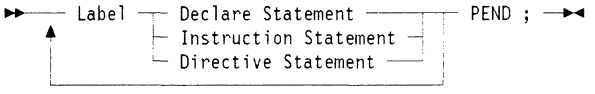
Program Attributes

The QPRCRTPG API creates programs that have the following attributes:

- An associated space of 0 bytes. The MODS MI instruction changes the size of this space.
- Observability.
- A blank extended attribute.

Program Syntax

A program object consists of an **instruction stream** and an **object definition table (ODT)**. The intermediate representation of a program defines both of these components. It consists of one or more statements:



Instruction statements define MI instructions placed in the instruction stream. Declare statements define program objects placed in the ODT. Directive statements:

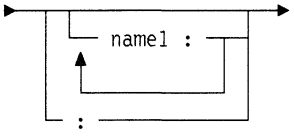
- Control the formatting of the output listing.
- Define entry-point program objects.
- Define symbolic breakpoints.
- Specify the end of the program.

The following sections explain how to define these statements.

Note: In the diagrams below, names that begin with an uppercase letter identify values specified in another diagram. Names that begin with a lowercase letter identify values defined in the table below the diagram.

Label

The following diagram and table show the possible labels:



Each name specified in the label generates a branch-point program object corresponding to the next MI instruction.

Constant	Range	Description
name1	Any	Label name for next instruction

Declare Statement

Declare statements define program data objects. All the declare statements in a program build the object definition table (ODT).

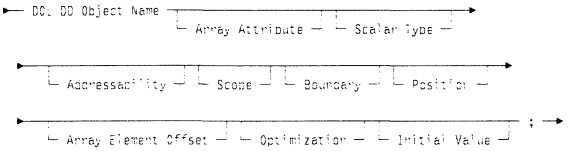
You cannot specifically declare branch and entry-point program objects. However, you can declare branch-point program objects using labels. You can also declare entry-point program objects using the entry directive statement.

The types of declare statements follow:

- Scalar Data Object Declare Statement
- Pointer Data Object Declare Statement
- Space Pointer Machine Object Declare Statement
- Operand List Declare Statement
- Instruction Definition List Declare Statement
- Exception Description Declare Statement
- Space Object Declare Statement
- Constant Object Declare Statement

Scalar-Data-Object Declare Statement

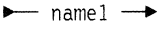
The following diagram and table show the scalar-data-object declare statement:



Only certain combinations of attributes are allowed based on the data object's addressability. The table below shows these combinations.

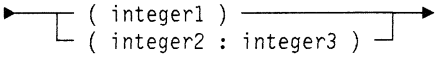
Address-ability	Array Attribute	Array Element Offset	Position	Boundary	Initial Value
STAT	X		X		X
	X			X	X
AUTO	X		X		X
	X			X	X
DEF	X		X	X	X
	X	X	X	X	
DIR	X		X	X	X
	X	X	X	X	
BAS	X		X	X	
BASPCO	X		X	X	
PARM	X			X	

Object Name: The following diagram and table show the possible object names:



Constant	Range	Description
name1	Any	Program object name to be declared

Array Attribute: The following diagram and table show the possible array attributes:

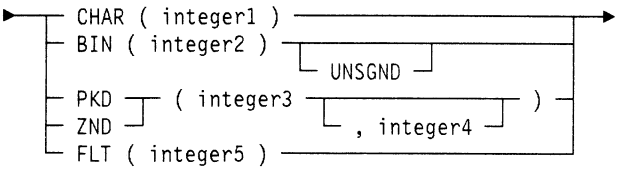


Constant	Range	Description
integer1	1 to 16 776 191	Dimension of the data object with an implied lower bound of 1.
integer2	-2 147 483 648 to 2 147 483 647	Lower bound of the array.
integer3	integer2 to 2 147 483 647	Upper bound of the array. The dimension (integer3 - integer2) cannot exceed 16 776 191.

Example: The following declare statements each define an array of 50 elements. The elements of ARRAY1 are numbered 1 to 50. The elements of ARRAY2 are numbered 0 to 49. Each element of the array is a BIN(2) field. The addressability of the arrays is static.

```
DCL DD ARRAY1(50) BIN(2);
DCL DD ARRAY2(0:49) BIN(2);
```

Scalar Type: The following diagram and tables show the possible data types of scalar items:

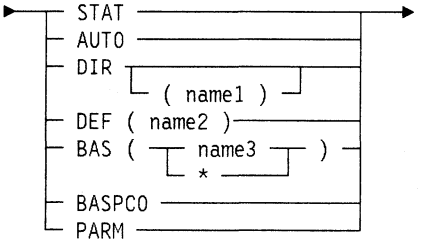


If you specify no value, the system uses BIN(2).

Keyword	Description
CHAR	Scalar type is a character string.
BIN	Scalar type is binary.
UNSGND	Scalar type is unsigned binary. If you do not specify this value, the scalar type is signed binary.
PKD	Scalar type is packed decimal.
ZND	Scalar type is zoned decimal.
FLT	Scalar type is floating-point.

Constant	Range	Description
integer1	See description.	Length in bytes of the character data object. If the data object is an array, the range is 1 to 32 767. Otherwise, the range is 1 to 16 776 191.
integer2	2 or 4	Length in bytes of the binary data object.
integer3	1 to 31	Total digits in the data object.
integer4	0 to integer3	Number of digits to the right of the assumed decimal point in the data object.
integer5	4 or 8	Precision in bytes of the data object.

Addressability: The following diagram and tables show the possible addressabilities:

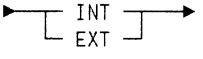


Keyword	Description
STAT	Addressability type is direct static.
AUTO	Addressability type is direct automatic.
DIR	Addressability type is defined. See "Using Space Objects" on page 24-15 for more information.
DEF	Addressability type is direct on the previous space.
BAS	Addressability type is based.
*	Object does not have explicit basing object.
BASPCO	Addressability type is based on process communication object space pointer.
PARM	Addressability type is a parameter.

Constant	Range	Description
name1	Any	Space object name
name2	Any	Scalar data object name or pointer data object name
name3	Any	Pointer data object name or space pointer object name

If you specify no value, the system uses STAT.

Scope: **Scope** refers to the ability to export a variable so that other programs can access it. The following diagram and table show the possible scopes:



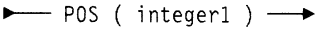
Keyword	Description
INT	Data object is not externally accessible
EXT	Data object is externally accessible

If you specify no value, the system uses INT.

Boundary: The following diagram and table show the possible boundaries:

Constant	Range	Description
integer1	1, 2, 4, 8, 16	Data object boundary

Position: The following diagram and table show the possible positions:



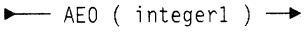
Constant	Range	Description
integer1	1 to 16 776 191	Data object position

Example: The following declare statements show how POS can be used along with DEF to access the same storage space in different ways:

```
DCL DD DATETIME CHAR(12);
DCL DD DATE CHAR(6) DEF(DATETIME);
DCL DD TIME CHAR(6) DEF(DATETIME) POS(7);
```

DATETIME represents a 12 character time and date stamp. The first 6 characters contain the date and the second 6 characters contain the time.

Array Element Offset: The following diagram and table show the possible array element offsets:

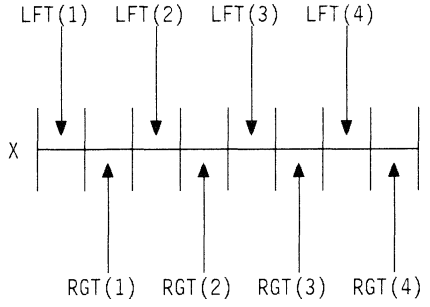


Constant	Range	Description
integer1	1 to 32767	Array element offset

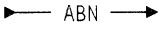
Example: The following example shows AEO used in conjunction with DEF and POS:

```
DCL DD X CHAR(16);
DCL DD LFT(4) BIN(2) DEF(X) AEO(4) POS(1);
DCL DD RGT(4) BIN(2) DEF(X) AEO(4) POS(3);
```

Both LFT and RGT redefine the storage declared by X. Because the size of each array element is smaller than the array element offset, there are 2-byte gaps between each array element:

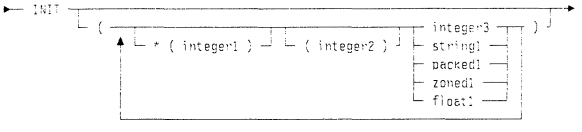


Optimization: **Optimization** determines whether or not an item can be moved to a register and stored there over time. The following diagram shows the possible optimization:



This value indicates that the data object contains an abnormal value. You cannot optimize the value for more than a single reference because the value may be changed in a manner that the QPRCRTPG API cannot detect.

Initial Value: The following diagram and table show each possible initial value:



Constant	Range	Description
integer1	1 to 16 776 191	Position of elements in a character string
integer1	-2 147 483 648 to 2 147 483 647	Position of elements in an array
integer2	1 to 16 776 191	Replication factor in a character string or array
integer3	Any	Initial value for signed and unsigned binary data objects
string1	Any	Initial value for character string data objects
packed1	Any	Initial value for packed decimal data objects
zoned1	Any	Initial value for zoned decimal data objects
float1	Any	Initial value for floating-point data objects

Example: The following declare statement declares and initializes a 10-element array:

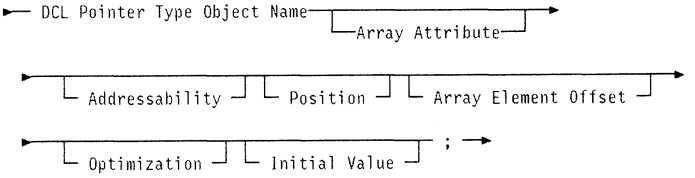
```
DCL DD IV(10) BIN(2) STAT INIT((1)10,*(2)(2)11,*(4)(3)12,*(7)(4)13);
```

There are four initial value elements. The following table describes this function:

Initial Value Element	Result	Position	Replication Factor	Initial value
(1)10	IV(1)=10	1 (default)	1	10
*(2)(2)11	IV(2)=11 IV(3)=11	2	2	11
*(4)(3)12	IV(4)=12 IV(5)=12 IV(6)=12	4	3	12
*(7)(4)13	IV(7)=13 IV(8)=13 IV(9)=13 IV(10)=13	7	4	13

Pointer-Data-Object Declare Statement

The following diagram and table show the pointer-data-object declare statement:

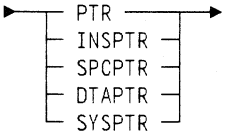


The system only allows certain combinations of attributes based on the data object's addressability. These combinations are listed as follows:

Addressability	Array Attribute	Array Element Offset Attribute	Position Attribute	Initial Value Attribute
STAT	X		X	X
AUTO	X		X	X
DEF	X	X	X	
			X	X

Address-ability	Array Attribute	Array Element Offset Attribute	Position Attribute	Initial Value Attribute
DIR	X	X	X	
			X	X
BAS	X		X	
BASPCO	X		X	
PARM	X			

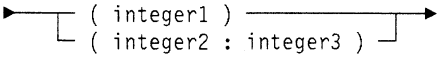
Pointer Type: The following diagram and table show the possible pointer types:



Keyword	Description
PTR	Pointer type is not specified.
INSPTR	Pointer type is the instruction pointer.
SPCPTR	Pointer type is the space pointer.
DTAPTR	Pointer type is the data pointer.
SYSPTR	Pointer type is the system pointer.

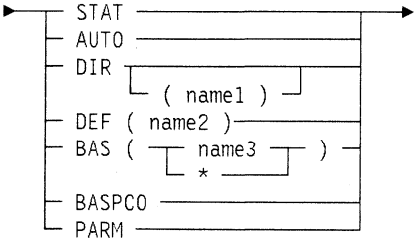
If you specify an initial value, you must specify INSPTR, SPCPTR, DTAPTR or SYSPTR.

Array Attribute: The following diagram and table show the possible array attributes:



Constant	Range	Description
integer1	1 to 1 000 000	Dimension of the data object with an implied lower bound of 1.
integer2	-2 147 483 648 to 2 147 483 647	Lower bound of the array.
integer3	integer2 to 2 147 483 647	Upper bound of the array. The dimension (integer3 - integer2) should not exceed 1 000 000.

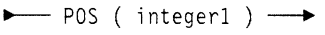
Addressability: The following diagram and tables show the possible addressabilities:



Keyword	Description
STAT	Addressability type is direct static.
AUTO	Addressability type is direct automatic.
DIR	Addressability type is defined. See "Using Space Objects" on page 24-15 for more information.
DEF	Addressability type is defined.
BAS	Addressability type is based.
*	Object does not have explicit basing object.
BASPCO	Addressability type is based on the process communication object space pointer.
PARM	Addressability type is parameter.

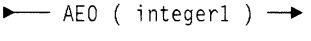
Constant	Range	Description
name1	Any	Space object name
name2	Any	Scalar data object name or the pointer data object name
name3	Any	Pointer data object name or the space pointer machine object name

Position: The following diagram and table show the possible positions:



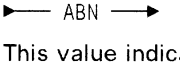
Constant	Range	Description
integer1	1 to 16 776 191	Data object position

Array Element Offset Value: The following diagram and table show the possible array element offset values:



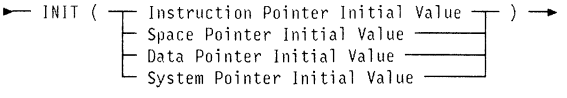
Constant	Range	Description
integer1	1 to 32 767	Array element offset

Optimization: The following diagram shows the possible optimizations:



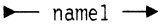
This value indicates that the data object contains an abnormal value. The system cannot optimize a value for more than a single reference because the value may be changed in a manner the QPRCRTPG API cannot find.

Initial Value: The following diagram shows each possible initial value:



An initial value can only be specified if a pointer-type value other than PTR is specified. The syntax of the initial value is based on the pointer-type value that was used.

Instruction Pointer Initial Value: The following diagram and table show the possible initial value for the instruction pointer:

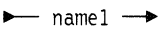


Constant	Range	Description
name1	Any	Label name

Example: The following statement declares and initializes an instruction pointer:

```
LABELI:

  :
  :

  DCL INSPTR INSTRUCTION_PTR INIT(LABELI);
```

Space Pointer Initial Value: The following diagram and table show the initial value for the space pointer:



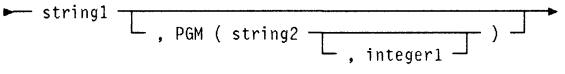
Constant	Range	Description
name1	Any	Scalar data object name or pointer data object name

Example: The following statement declares and initializes a space pointer:

```
DCL PTR ANY_POINTER;
DCL SPCPTR SPACE_PTR INIT(ANY_POINTER);
```

The pointer SPACE_PTR is initialized to point to the space location containing ANY_POINTER. It does *not* contain the value of ANY_POINTER.

Data Pointer Initial Value: The following diagram and table show the initial value for the data pointer:



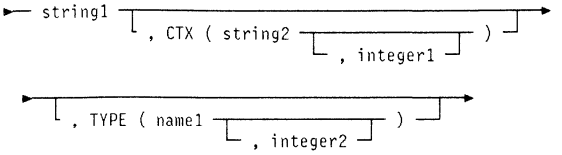
Constant	Range	Description
string1	32 bytes	External data object name
string2	30 bytes	Program containing external data object
integer1	0 to 255	Subtype of program

Example: The following statement declares and initializes a data pointer:

```
DCL DTAPTR DVALUE INIT("DBINARY",PGM("DPGM"));
```

The pointer DTAPTR refers to the externally defined program object DBINARY contained in program DPGM.

System Pointer Initial Value: The following diagram and tables show the initial value for the system pointer:



Constant	Range	Description
string1	1 to 30 bytes	System object
string2	1 to 30 bytes	Context where the system object is located
integer1	0 to 255	Subtype of the context
name1	See table below.	Symbolic type of the system object
integer2	0 to 255	Subtype of the system object

The following system object types are supported:

Type	Description
PGM	Program
CTX	Context
Q	Queue
SPC	Space
PCS	Process control space

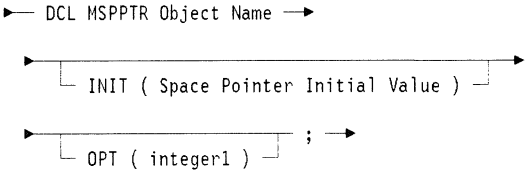
Example: The following statement declares and initializes a system pointer:

```
DCL SYSPTR SYSTEM_PTR INIT("MYPGM",CTX("PGMLIB"),TYPE(PGM));
```

The pointer SYSTEM_PTR refers to the *PGM object MYPGM in the PGMLIB library.

Space-Pointer-Machine-Object Declare Statement

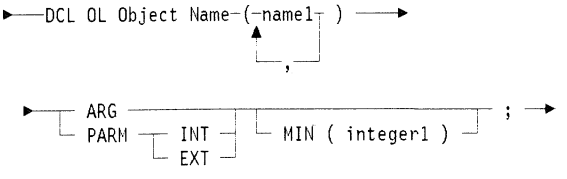
The following diagram and table show the space-pointer-machine-object declare statement:



Constant	Range	Description
integer1	0 to 255	Optimization priority value

Operand-List Declare Statement

The following diagram and tables show the operand-list declare statement:



Keyword	Description
ARG	Defines the argument list
PARM	Defines the parameter list
INT	An internal parameter list
EXT	An external parameter list

Constant	Range	Description
name1	Any	Scalar data object or a pointer data object name. Up to 255 names can be specified.
integer1	0 to 255	Minimum number of elements that the list can contain. This implicitly defines a variable-length operand list. If you do not specify the operand list, the system defines a fixed-length operand list. Up to 255 names can be specified.

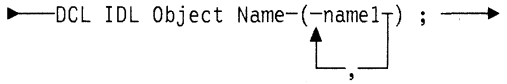
Example: The following statements declare both argument and parameter operand lists along with the associated argument and parameter data objects:

```
DCL DD ARG1 BIN(2);
DCL DD ARG2 CHAR(3);
DCL OL ARGUMENT_LIST (ARG1, ARG2) ARG;

DCL DD PARM1 BIN(2) PARM;
DCL DD PARM2 CHAR(3) PARM;
DCL OL PARAMETER_LIST (PARM1, PARM2) PARM EXT;
```

A parameter operand list that refers to the data objects has parameter (PARM) addressability.

Instruction-Definition-List Declare Statement

The following diagram and table show the instruction-definition-list declare statement:



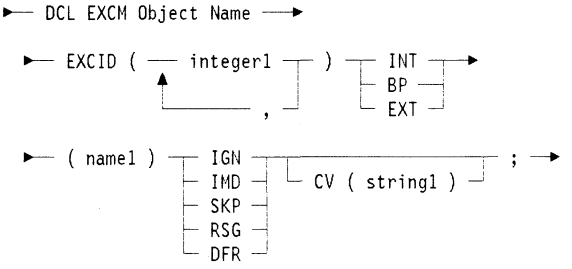
Constant	Range	Description
name1	Any	Label name. Up to 255 names can be specified.

Example: The following statements declare and use an instruction definition list:

```
LABEL1:

  :
  :

  DCL IDL INSTRUCTION_LIST (LABEL1,LABEL2,LABEL3);

  :
  :

LABEL2:
  B INSTRUCTION_LIST(3); /* Branch to LABEL3 */

  :
  :

LABEL3:
```

Exception-Description Declare Statement

The following diagram and tables show the exception-description declare statement:



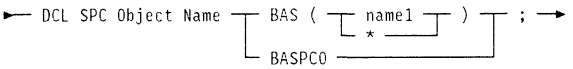
Keyword	Description
INT	Exception handler type is the internal entry point.
BP	Exception handler type is the internal branch point.
EXT	Exception handler type is the external entry point.
IGN	Exception handling action ignores any exceptions and continues processing.
IMD	Exception handling action passes control to the specified exception handler.
SKP	Exception handling action is to continue to search for another exception description to handle the exception.
RSG	Exception handling action continues to search for an exception description by signaling the exception again to the previous call.
DFR	Exception handling action postpones handling and saves exception data for later exception handling.

Constant	Range	Description
integer1	0 to 65535	Exception identifier

Constant	Range	Description
name1	Any	Name of the label for branch point exception handlers, name of the entry point for the internal exception handlers, and the name of the system pointer for the external exception handlers
string1	1 to 32 bytes	Compare value

Space-Object Declare Statement

The following diagram and tables show the space-object declare statement:



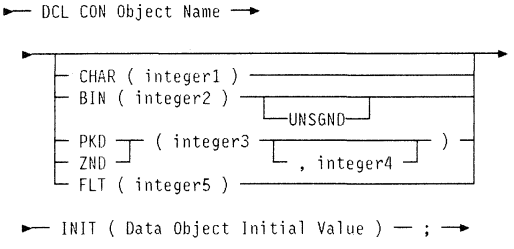
Keyword	Description
BAS	Addressability type is based.
*	Object does not have explicit basing object.
BASPCO	Addressability type is based on process communication object space pointer.

Constant	Range	Description
name1	Any	Basing pointer name for the space

For information on using space objects, refer to "Using Space Objects" on page 24-15.

Constant-Object Declare Statement

The following diagram and tables show the constant-object declare statement:



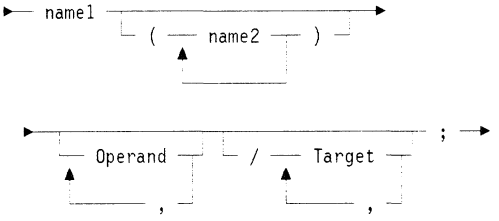
Keyword	Description
CHAR	Scalar type is character string.
BIN	Scalar type is binary.
UNSGND	Scalar type is unsigned binary.
PKD	Scalar type is packed decimal.
ZND	Scalar type is zoned decimal.
FLT	Scalar type is floating-point.

Constant	Range	Description
integer1	1 to 32 767	Length in bytes of the character data object
integer2	2 or 4	Length in bytes of the binary data object
integer3	1 to 31	Number of decimal digits
integer4	0 to integer3	Number of fractional digits
integer5	4 or 8	Number of bytes in floating-point constant

If you do not specify a scalar type, the system uses BIN(2).

Instruction Statement

An instruction statement defines an MI instruction. The instruction stream used to create the program is made up of all the instruction statements in the intermediate representation of the program.



Constant	Range	Description
name1	See description.	Opcode for this instruction, as defined in the *MI Functional Reference*.
name2	S, R, B, I	This is the form of the instruction. **S** Short **R** Round **B** Branch **I** Indicator

For the semantic meanings and the syntax restrictions (number and types of operands, optional forms, and so on) for individual MI instructions, see the *MI Functional Reference*.

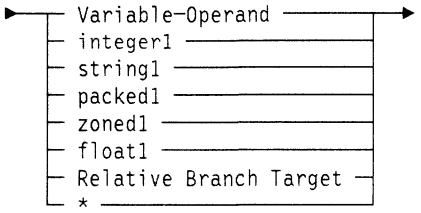
Following the abbreviated instruction name, you can specify the optional forms of certain MI instructions using a string of characters enclosed in parentheses. The following is an example of some of the various combinations possible for a single MI instruction, ADD NUMERIC:

```
ADDN      A,B,C;                Add numeric (A=B+C)
ADDN(S)   A,B;                  Add numeric short (A=A+B)
ADDN(SR)  A,B;                  Add numeric short and round (A=A+B)
ADDN(SB)  A,B/POS(X),NEG(Y);    Add numeric short and branch  (A=A+B,
                                branch to X if A>0, branch to Y if A<0)
ADDN(RI)  A,B,C/POS(I),NEG(J); Add numeric round and indicator (A=B+C;
                                I='on' if A>0; j='on' if A<0 )
```

Also note that the order of characters in the optional form string is not significant. Thus, all of the following instructions are both valid and equivalent:

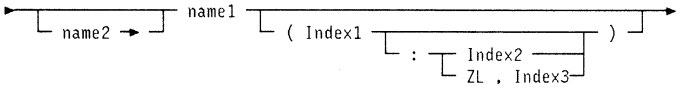
```
ADDN(SRB)A,B/POS(X);     Add numeric short, round and branch
ADDN(SBR)A,B/POS(X);     Add numeric short, round and branch
ADDN(RSB)A,B/POS(X);     Add numeric short, round and branch
```

Operand: The following diagram and table show the possible operands:



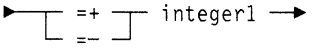
Constant	Range	Description
integer1	Any	Numeric binary scalar operand
string1	Any	Character scalar operand
packed1	Any	Numeric packed decimal scalar operand
zoned1	Any	Numeric zoned decimal scalar operand
float1	Any	Numeric floating-point scalar operand (4 or 8 bytes)
*		Null operand

Variable Operand: The following diagram and table show the possible variable operands:



Constant	Range	Description
name1	Any	Data object name to be used as a primary operand.
name2	Any	Pointer data object to be used as the basing pointer.
Index1	See description.	Subscript or substring start position. The range for array subscripts is between the lower bound of the array and the upper bound of the array. The range for substrings is between 1 and 16 776 191.
Index2	1 to 32 767	Length of the substring.
Index3	0 to 32 767	Length of the substring (zero allowed).

Relative Branch Target: The following diagram and table show the possible relative branch targets:



Constant	Range	Description
integer1	1 to 4095	Branch target instruction number *relative* to the current instruction. You must label the target (named or null label).

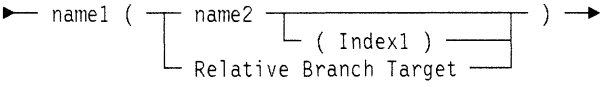
Note: You cannot use blanks between either the '=+' symbol set and integer1 or the '=-' symbol set and integer1. However, a blank must precede the symbol sets.

Example: The following instructions illustrate the use of relative branch targets:

```
  CPYNV X,0;
  CMPBLA(B) A,'1'/EQ( =+2);
  CPYNV X,1;
: CPYNV Y,X;        /* Destination of relative branch */
```

Note: A null label is placed in the destination instruction of the relative branch.

Target: The following diagram and table show the possible targets:



Constant	Range	Description
name1	See keyword table.	Keyword for branch or indicator forms. You can use an N before keywords to negate the condition except for IGN and DFR. See "Resultant Conditions", under each MI instruction for the valid values.
name2	Any	Label name, instruction pointer name, or instruction definition list name for the branch form. The name of character variable is for the indicator form.
Index1	1 to 255	Instruction definition list index. You can only specify this value when name2 is the name of an instruction definition list.

Notice that a null label has been placed on the destination instruction of the relative branch.

The following table shows the branch and indicator keywords:

Keyword	Description
Group 1	

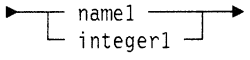
Keyword	Description
HI	High
MXD	Mixed
NOR	Normalized
POS	Positive
TR	Truncated record
ZC	Zero and carry
Group 2	
CR	Complete record
DEN	Denormalized
IGN	Exception ignored
LO	Low
NEG	Negative
NTZNTC	Not-zero and no carry
RO	Receiver overrun
Group 3	
AUTH	Authorized
DFR	Exception postponed
DQ	Dequeued
EQ	Equal
INF	Infinity
SE	Source all used
SGN	Signaled
ZER	Zero
ZNTC	Zero and no carry
Group 4	
EC	Escape code encountered
NAN	Not a number (NaN)
NTZC	Not-zero and carry
UNEQ	Unequal
UNOR	Unordered

By adding N to the beginning of the appropriate keyword you can form a not condition. For example, the code for "not equal" is NEQ.

All conditions coded on a particular instruction must be mutually exclusive. All conditions within a group are equivalent, and therefore, only one may be specified. For example, POS (positive) and HI (high) cannot be coded on the same instruction.

The not form of a condition is satisfied by any condition from another group. For example, NEQ (not equal) is satisfied by HI (high), LO (low), or UNOR (unordered). Therefore, you cannot specify NEQ with any of the other three. However, you can use NEQ and EQ (or any other keyword in group 3) together because they are mutually exclusive.

Index: The following diagram and table show the possible indexes:



Constant	Range	Description
name1	See description below.	Binary variable to use as the index
integer1	See description below.	Integer value to use as the index

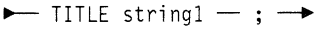
An index is a numeric value that qualifies an array or substring reference. The context in which the index is used determines the range. For more information, refer to the preceding tables.

Directive Statements

The directive statements are as follows:

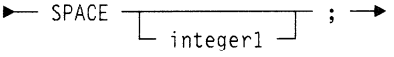
- Title Directive Statement
- Space Directive Statement
- Eject Directive Statement
- Break Directive Statement
- Entry Directive Statement
- Reset Directive Statement
- Program End Directive Statement

Title Directive Statement: The title directive statement causes a heading to appear on the listings. Only one title directive statement may be specified in a program. The following diagram and table show the title directive statement:



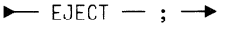
Constant	Range	Description
string1	Any	Text of the title

Space Directive Statement: The space directive statement causes a blank line to appear in the listing. The following diagram and table show the space directive statement:

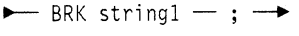


Constant	Range	Description
integer1	Any	Number of lines to skip

Eject Directive Statement: The eject directive statement causes the next line to appear on a new page. The following diagram shows the eject directive statement:

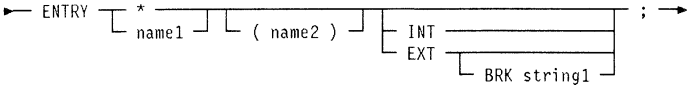


Break Directive Statement: The break directive statement allows symbolic breakpoints to be defined. The following diagram and table show the break directive statement:



Constant	Range	Description
string1	Any	Breakpoint name

Entry Directive Statement: The following diagram and tables show the entry directive statement:



Keyword	Description
INT	Internal entry point.

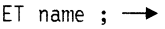
Keyword	Description
EXT	External entry point.
BRK	Symbolic breakpoint is associated with the entry point.
*	Entry point defined has no name or is associated with the next MI instruction.

Constant	Range	Description
name1	Any	Entry point name being defined
name2	Any	Parameter list name for this entry point
string1	1-10 bytes	Breakpoint name

The default scope is internal (INT).

The entry statement defines entry point program objects. The next instruction number is associated with this entry point. The entry statement is to be the definition point for this object, so the ODT number assigned to this object is the next available ODT number.

Reset Directive Statement: The following diagram shows the reset directive statement:



The specified name is a previously declared space object. The reset statement causes subsequent data object declarations containing the DIR attribute to use the specified space object. The system maintains next byte counts for each space object; these counts are not affected by the reset statement. For more information, see "Using Space Objects."

Program End Directive Statement: The following diagram shows the program end directive statement:

This must be the last statement in the program. To ensure comments and strings end before processing the PEND statement, use the following statement:

```
/*'/*'/*"/*"*/; PEND;;;
```

Coding Techniques

This section contains additional information for coding the intermediate representation of a program.

Using Declare Statements

Use the following guidelines when using declare statements:

- A declare statement for data objects defined on another data object must occur after the declare statement for the data object on which it is defined.

  Example: The following sets of declare statements are valid:

```
DCL DD A CHAR(5);
DCL DD B CHAR(1) DEF(A);
```

```
DCL DD A CHAR(5);
DCL DD X BIN(2);
DCL PTR P1 AUTO;
DCL DD B CHAR(1) DEF(A);
```

  Example: The following declare statements are not valid because B is defined on A but is declared before A:

```
DCL DD B CHAR(1) DEF(A);
DCL DD A CHAR(5);
```

  This restriction also applies when there is a chain of dependencies.

  Example: In the figure below, B is defined on A and C is defined on B:

```
DCL DD A CHAR(5);
DCL DD B CHAR(3) DEF(A);
DCL DD C CHAR(1) DEF(B);
```

  If any object in a chain of definitions, as shown in the previous examples, has an initial value specified, then the following restrictions apply:

  - No object in that chain can have the BAS (based) addressability attribute.
  - The highest level data object in the chain must be either static or automatic.
  - When you initialize the same area twice, the system uses the last value.

  Example: The following declare statements are valid because:

  - The BAS addressability attribute is not used.
  - Data object A (implicitly) has the static addressability attribute.

```
DCL DD A CHAR(5);
DCL DD B CHAR(3) DEF(A) INIT(C'YES');
DCL DD C CHAR(1) DEF(B);
```

- All declare statements for the objects that make up the elements of an operand list must precede the declare statement for the operand list.
- When a declare statement for an exception description refers to a system pointer, the declare statement for the system pointer must precede the DCL for the exception description.

Using Space Objects

Space objects, when used in conjunction with program objects declared with the DIR attribute, provide a convenient way of declaring structures.

Note: Space objects, as used here, do not refer to OS/400 space objects.

When you declare a space object, a scalar data object with a scalar type of CHAR(32767) is created. This object contains the structure to be defined. Associated with this object is a "next byte" count. This value is initially 1 and

represents the position where the next structure element will be placed.

Example: Simple Space Objects: After you declare a space object, you can declare one or more scalar or pointer data objects with an addressability attribute of DIR. As a result, the system automatically declares each object with the DEF and POS attributes. The name associated with the DEF attribute is the most recently declared space object. The value associated with the POS attribute is the space object's next byte count. After you declare the object, the system sets the next byte count associated with the space object to the next available position within the structure.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPC X BAS(PTR);          DCL DD X CHAR(32767) BAS(PTR);
DCL DD A CHAR(2) DIR;        DCL DD A CHAR(2) DEF(X) POS(1);
DCL DD B ZND(5,2) DIR;       DCL DD B ZND(5,2) DEF(X) POS(3);
DCL DD C FLT(4) DIR;         DCL DD C FLT(4) DEF(X) POS(8);
```

Example: Explicit Position Values: Data objects declared with DIR may also have an explicit POS value. The object is defined on the appropriate space object and uses the specified POS value. However, the next byte count is changed only if the POS value causes the count to increase.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPC X BAS(PTR);              DCL DD X CHAR(32767) BAS(PTR);
DCL DD A CHAR(4) DIR;            DCL DD A CHAR(4) DEF(X) POS(1);
DCL DD B CHAR(4) POS(20) DIR;    DCL DD B CHAR(4) DEF(X) POS(20);
DCL DD C CHAR(4) DIR;            DCL DD C CHAR(4) DEF(X) POS(24);
DCL DD D CHAR(4) POS(10) DIR;    DCL DD D CHAR(4) DEF(X) POS(10);
DCL DD E CHAR(4) DIR;            DCL DD E CHAR(4) DEF(X) POS(28);
```

Example: Implicit Boundary Alignment: When you declare objects with an explicit or implicit boundary other than 1, the object is declared after the next byte position in order to ensure boundary alignment. Space objects are assumed to begin on a 16-byte boundary. You must ensure this condition exists at run-time.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPC X BAS(PTR);          DCL DD X CHAR(32767) BAS(PTR);
DCL DD A CHAR(1) DIR;        DCL DD A CHAR(1) DEF(X) POS(1);
DCL DD B FLT(4) DIR;         DCL DD B FLT(4) DEF(X) POS(5);
DCL PTR C DIR;               DCL PTR C DEF(X) POS(17);
```

Example: Reset Directive Statement: You can use the reset directive statement to change the name of the space object to be used by subsequent declare statements.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPCPTR PTR1;             DCL SPCPTR PTR1;
DCL SPCPTR PTR2;             DCL SPCPTR PTR2;

DCL SPC X BAS(PTR1);         DCL DD X CHAR(32767) BAS(PTR1);
DCL DD A CHAR(2) DIR;        DCL DD A CHAR(2) DEF(X) POS(1);
DCL DD B ZND(5,2) DIR;       DCL DD B ZND(5,2) DEF(X) POS(3);

DCL SPC Y BAS(PTR2);         DCL DD Y CHAR(32767) BAS(PTR2);
DCL DD C CHAR(5) DIR;        DCL DD C CHAR(5) DEF(Y) POS(1);
DCL DD D CHAR(7) DIR;        DCL DD D CHAR(7) DEF(Y) POS(6);

RESET X;
DCL DD E CHAR(3) DIR;        DCL DD E CHAR(3) DEF(X) POS(8);
```

Constants

This section describes the syntax of constant values.

Integer: Integers define signed and unsigned binary scalar data values. The two forms of integers are decimal and hexadecimal. The decimal form is a sequence of digits optionally preceded by a sign. The hexadecimal form is a string of hexadecimal digits delimited with apostrophes and preceded by an H. Neither form may exceed the 4-byte limit on binary numbers. When the value of the integer is between -4095 and +8191, the QPRCRTPG API converts the integer to an immediate operand where it can.

Example:

```
+123
-1
54788

H'0F0D'
H'0123'
H'5E2D1AB4'
```

String: Strings define scalar character string data values. The three types of string constants are character form, hexadecimal form, and Hollerith form.

The character form is a delimited string optionally preceded by a C. Apostrophes or double quotation marks may be used for this form. The hexadecimal form is a delimited string of hexadecimal digits preceded by an X. The Hollerith form is a string of bytes preceded by the count of the number of bytes in the string. The syntax is:

```
< count | string >
```

The count in the preceding syntax is the number of characters in the string. The QPRCRTPG API ensures that the string contains the right number of characters by checking for the > character. No blanks are allowed between < and > unless they are part of the string. The QPRCRTPG API simply flags the constant as in error if the right corner bracket does not appear in the correct position.

Example: The following groups of strings are equivalent:

```
'ABCDE'
C'ABCDE'
X'C1C2C3C4C5'
<5|ABCDE>

'TE''ST'
"TE'ST"
X'E3C57DE2E3'
<5|TE'ST>

'/*'
X'615C'
<2|/*>
```

Packed: Packed constants define packed decimal scalar data values. Packed constants are a string of decimal digits delimited with apostrophes. They can have an embedded decimal point and can be preceded by a sign. P must precede the delimited string. Packed constants have a maximum of 31 significant digits.

Note: You must specify at least one numeric digit.

Example:

```
P'+123.456'
P'1'
P'-1'
P'-123.345345345345'
P'+.00000000000001'
```

Zoned: Zoned constants define zoned decimal scalar data values. The external representation of zoned constants is the same as that for packed constants except that the preceding character is a Z.

Note: You must specify at least one numeric digit.

Example:

```
Z'+123.456'
Z'1'
Z'-1'
Z'-123.345345345345'
Z'+.00000000000001'
```

Floating-Point Constants: Floating-point constants define floating-point scalar data values. You must specify whether the constant is a 4-byte (short floating-point) or an 8-byte (long floating-point) value.

There are two ways to represent floating-point values. First, you can specify floating-point constants as a delimited string of decimal digits possibly with an embedded decimal point and optionally preceded by a sign. An F for short floating-point values or an E for long floating-point values must precede the delimited string. An E in the string determines the start of the base 10 exponent. You specify the exponent as signed.

Second, you can specify floating-point constants as a string of hexadecimal digits. The delimited string must be preceded by an XF for short floating-point values or an XE for long floating-point values.

Note: You must specify at least one numeric digit.

Example:

Short Floating-Point Values	**Long Floating-Point Values**
F'0'	E'0'
F'+12'	E'+12'
F'-12.21'	E'-12.21'
F'12.34E2'	E'12.34E2'
F'+3.2345678E-02'	E'+3.2345678E-02'
XF'449A4000'	XE'46CE6F37FFBE8722'
XF'40490FD0'	XE'400921F9F01B866E'

Several special values are allowed:

Short Floating-Point Values	**Long Floating-Point Values**	
F'MNAN'	E'MNAN'	Masked Not A Number
F'UNAN'	E'UNAN'	Unmasked Not A Number
F'+INF'	E'+INF'	Plus Infinity
F'-INF'	E'-INF'	Minus Infinity

Note: You must use floating-point constants to initialize floating-point data objects.

Name: Names specified in the intermediate representation of a program are a sequence of characters of up to 48 characters in length. You cannot use the following characters as the first character of the name:

```
blank /,;():<+'"%-0123456789
```

You cannot use the following characters in subsequent characters of the name:

```
blank /,;():<+'"%
```

Example:

```
.NAME
NAME
THIS_IS_A_NAME
THIS_IS_A_NAME_2
&NAME
!NAME
?NAME
.0001
```

Note: Symbols that begin with a period (.) are not inserted into the program's symbol table and may not be referred to by the OS/400 debug function.

Comments: Comments, in the intermediate representation of a program, may appear anywhere in the text. Comments are treated as blanks so they are significant in finding tokens. Comments are a string of characters starting with /* and ending with */. If a comment occurs immediately following a semicolon, it prints as a separate line (or a multiple line as required) on the listing. If a comment is embedded in a statement, then it appears as a part of that statement, such as a remark.

Example: The following statements are equivalent:

```
CPYBLA A,B;
CPYBLA A, /* C-> */ B ;
CPYBLA A,B;  /* B is based on C */
```

Blanks: You can use strings of blanks of any length in the intermediate representation of a program. Blanks act as delimiters in finding tokens and in some places are necessary as in separating the opcode and operand in an instruction statement.

Example: The following statements are equivalent:

```
ADDN A,B,C;
ADDN        A    ,     B              , C             ;
```

Retrieve Command Information (QCDRCMDI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified command name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Command Information (QCDRCMDI) API retrieves information from a command definition object and places it into a single variable in the calling program. The amount of information returned depends on the size of the variable. The information returned is the same information returned by the Display Command (DSPCMD) command.

You can use the QCDRCMDI API to retrieve any operable command. This includes both interactive (such as the Display Program (DSPPGM), Create Library (CRTLIB), and so on) and non-interactive (such as DO, IF, ELSE, and so on) commands. It does not include command definition statements that appear in command source, such as CMD, DEP, ELEM, PARM, PARMCTL, and QUAL.

Authorities and Locks

Command Definition Object Authority *USE
Library Authority *USE
Command Definition Object Lock *SHRRD

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. The maximum size for this area is 8 bytes. You can specify the size of this area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If this value is larger than the actual size of the receiver variable, the result may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the command information to be returned. You can use this format:

CMDI0100 Basic command information.
CMDI0200 Complete command information.

Qualified command name
INPUT; CHAR(20)
The name of the command whose values are to be retrieved. The first 10 characters contain the name of the command, and the second 10 characters contain the name of the library where the command is located.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

CMDI0100 Format

The following table describes the information that is returned in the receiver variable for the CMDI0100 format. For detailed descriptions of the fields, see "Field Descriptions" on page 24-19.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Command name
18	12	CHAR(10)	Command library name
28	1C	CHAR(10)	Command processing program name
38	26	CHAR(10)	Command processing program library name
48	30	CHAR(10)	Source file name
58	3A	CHAR(10)	Source file library name
68	44	CHAR(10)	Source file member name
78	4E	CHAR(10)	Validity check program name
88	58	CHAR(10)	Validity check program library name

Offset		Type	Field
Dec	Hex		
98	62	CHAR(10)	Mode information The characters of this field are as follows and they can have a value of '0' (does not apply) or '1' (does apply): **1** Production mode **2** Debug mode **3** Service mode **4-10** Reserved
108	6C	CHAR(15)	Where allowed to run The characters of this field are as follows and they can have a value of '0' (does not apply) or '1' (does apply): **1** Batch program (*BPGM) **2** Interactive program (*IPGM) **3** Can be run using QCMDEXC or QCAEXEC (*EXEC) **4** Interactive job (*INTERACT) **5** Batch job (*BATCH) **6** Batch REXX procedure (*BREXX) **7** Interactive REXX procedure (*IREXX) **8-15** Reserved
123	7B	CHAR(1)	Allow limited user
124	7C	BINARY(4)	Maximum positional parameters
128	80	CHAR(10)	Prompt message file name
138	8A	CHAR(10)	Prompt message file library name
148	94	CHAR(10)	Message file name
158	9E	CHAR(10)	Message file library name
168	A8	CHAR(10)	Help panel group name
178	B2	CHAR(10)	Help panel group library name
188	BC	CHAR(10)	Help identifier
198	C6	CHAR(10)	Search index name
208	D0	CHAR(10)	Search index library name
218	DA	CHAR(10)	Current library
228	E4	CHAR(10)	Product library
238	EE	CHAR(10)	Prompt override program name
248	F8	CHAR(10)	Prompt override program library name
258	102	CHAR(6)	Restricted to target release
264	108	CHAR(50)	Text description
314	13A	CHAR(36)	Reserved

CMDI0200 Format

The following table describes the information that is returned in the receiver variable for the CMDI0200 format. For detailed descriptions of the fields, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CMDI0100
350	15E	CHAR(10)	REXX source file name
360	168	CHAR(10)	REXX source file library name
370	172	CHAR(10)	REXX source file member name
380	17C	CHAR(10)	REXX command environment name
390	186	CHAR(10)	REXX command environment library name
400	190	CHAR(40)	Reserved
440	1B8	BINARY(4)	Number of REXX exit entries
444	1BC	BINARY(4)	Length of a REXX exit entry
Format of a REXX exit entry (repeated by the number of REXX exit entries)			
See note	See note	CHAR(10)	REXX exit program name
See note	See note	CHAR(10)	REXX exit program library name
See note	See note	BINARY(4)	REXX exit code
Note: The decimal and hexadecimal offsets to the above 3 fields depend on the number of REXX exit entries and the length of a REXX exit entry. The REXX exit entry fields (currently REXX exit program name, REXX exit program library name, and REXX exit code) repeat, in the order listed, by the number of REXX exit entries defined for this command.			

Field Descriptions

For more information on the following fields, refer to the documentation for the Create Command (CRTCMD) command in the *CL Reference*.

Allow limited user. Whether or not a user with limited authorities is allowed to run this command. The possible values are 0 (*NO) or 1 (*YES).

Bytes available. The length of all data available for the requested format. All available data is returned if enough space is provided.

Bytes returned. The length of all data actually returned. If the data is truncated because the receiver variable is

not large enough to hold all of the data available, this value will be less than the bytes available.

Command library name. The name of the library in which the command description resides.

Command name. The name of the command description about which information is being returned.

Command processing program library name. The name of the library in which the process command program resides. This field will be blank if the command processing program name contains the special value *REXX.

Command processing program name. The name of the program that accepts parameters from the command and processes the command. The possible values are:

**REXX* The REXX fields returned in the CMDI0200 format contain valid information about the command.
CPP-program-name
The command processing program name.

Current library. The name of the library used as the current library during the processing of this command. The possible values are:

**NOCHG*
The current library does not change for the processing of this command. If the current library is changed during processing of the command, the change remains in effect after command processing is complete.
**CRTDFT*
No current library is active during processing of the command. The current library that was active before command processing began is restored when processing is completed.
library-name
The name of the library that is used as the current library. When command processing is completed, the current library is restored to its previous value.

Help identifier. The name of the general help module for the names of the help identifiers for this command. The possible values are:

**NONE* No help is available.
**CMD* The name of the command is used.
help-ID-name
A user-specified help module was used.

Help panel group library name. The name of the library in which the panel group resides.

Help panel group name. The name of the panel group in which the online help information exists for this command. If *NONE is returned, no help is available for this command.

Length of a REXX exit entry. The length of one REXX exit entry. This value is currently 24: 10 bytes for the REXX exit program name, 10 bytes for the REXX exit library name, and 4 bytes for the REXX exit code.

Maximum positional parameters. The maximum number of parameters than can be coded positionally for this command. The possible values are:

-1 No maximum positional coding limit was specified for this command.
0 through 75
The maximum number of parameters that can be coded positionally for this command.

Message file library name. The name of the library in which the message file resides.

Message file name. The message file from which messages identified on the DEP statements used to define the command are retrieved.

Mode information. The mode of operating environment to which the command applies. The valid values for the characters defined in this field are:

0 Does not apply.
1 Does apply.

Number of REXX exit entries. The number of times the REXX exit entries (consisting of 3 fields – REXX exit program name, REXX exit program library name, and REXX exit code) are repeated.

Product library. The name of the product library that is in effect during the processing of the command. The possible values are:

**NOCHG*
The product library does not change for the processing of this command.
**NONE* There is no product library in the job's library list.
library-name
The name of the library that is used as the product library during the processing of the command.

Prompt message file library name. The name of the library in which the prompt message file resides.

Prompt message file name. The name of the message file that contains the prompt text for this command. If *NONE is returned, no message file was specified for prompt text.

Prompt override program library name. The name of the library in which the prompt override program resides.

Prompt override program name. This is the name of the prompt override program that replaces (on the prompt display) the default values with the current actual values for the parameter. If *NONE is returned, no prompt override program was specified for this command.

Reserved. An ignored field.

Restricted to target release. The version, release, and modification level this command is restricted to. This applies only to a command used in a CL program and it must match the contents of the target release parameter on the CRTCLPGM (create CL program) command. See

the CRTCLPGM command for more information. This field has the format VvRrMm where:

Vv The character V is followed by a 1-character version number.
Rr The character R is followed by a 1-character release level.
Mm The character M is followed by a 1-character modification level.

If this field is blank, the command can be used in the current release.

REXX command environment library name. The name of the library in which the REXX command environment program resides.

REXX command environment name. The command environment program that is active when the REXX CPP starts to run. The REXX interpreter calls this program to process commands encountered in the REXX procedure. The possible values are:

**COMMAND*
The AS/400 system control language command environment is used.
**CPICOMM*
The Common Programming Interface (CPI) for Communications command environment is used. CPICOMM is the command environment used for CL commands that are embedded within a REXX procedure.
program-name
The name of the program to process commands found in the REXX procedure.

REXX exit code. A value which controls the conditions in which the REXX exit program is called. The possible values are:

2 The exit program is called whenever an external function or subroutine has been called by the REXX program. The exit program is responsible for locating and calling the requested routine.
3 The exit program is called whenever the interpreter is going to call a command. The exit program is responsible for locating and calling the command.
4 The exit program is called whenever a REXX instruction or function attempts an operation on the REXX external data queue.
5 The exit program is called when session input or output operations are attempted.
7 The exit program is called after running each clause of the REXX procedure to determine whether it must be stopped.
8 The exit program is called after running each clause of the REXX program to check if tracing must be turned on or off.
9 The exit program is called before interpretation of the first instruction of a REXX procedure.
10 The exit program is called after interpretation of the last instruction of a REXX procedure.

REXX exit program library name. The name of the library in which the REXX exit program resides.

REXX exit program name. The exit program used when the REXX interpreter is started under the conditions specified by the REXX exit code for this program.

REXX source file library name. The name of the library in which the REXX source file resides.

REXX source file member name. The name of the source file member that contains the REXX procedure that is the command processing program.

REXX source file name. The name of the source file that contains the REXX procedure that is the command processing program. The possible values are:

QREXSRC
The IBM-supplied source file, QREXSRC, contains the source member that is used.
source-file-name
The name of the REXX source file that is used.

Search index library name. The name of the library in which the help search index resides.

Search index name. The help search index to be used when the search index function key is pressed. The possible values are:

**SYSTEM*
The system index file, QHSS1.
**NONE* No help search index is used.
search-index-name
The name of the help search index that is used.

Source file library name. The name of the library in which the source file resides.

Source file member name. The name of the source file member that contains the command definition statements used to create the command.

Source file name. The name of the source file that contains the source file member used to create the command.

Text description. The user text, if any, used to briefly describe the command and its function.

Validity check program library name. The name of the library in which the validity checking program resides.

Validity check program name. The name of a program that performs additional user-defined validity checking on the parameters in the command. If *NONE is returned, no separate user-defined validity checking is done for this command; all validity checking is done by the command analyzer and the command processing program.

Where allowed to run. The environments in which this command is allowed to run. The valid values for the characters defined in this field are:

0 Does not apply.
1 Does apply.

Error Messages

| CPF2150 E Object information function failed.
| CPF2151 E Operation failed for &2 in &1 type *&3.
| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF3C24 E Length of the receiver variable is not valid.
| CPF6250 E Cannot display or retrieve command &1 in library &2.
| CPF8122 E &8 damage on library &4.
| CPF8123 E Damage on object information for library &4.
| CPF8129 E Program &4 in &9 damaged.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9806 E Cannot perform function for object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9811 E Program &1 in library &2 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9821 E Not authorized to program &1 in library &2.
| CPF9830 E Cannot assign library &1.

Retrieve Program Information (QCLRPGMI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Program and library name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Program Information (QCLRPGMI) API lets you retrieve program information and place it into a single variable in the calling program. The amount of information returned is limited to the size of the variable. This information is the same as the information returned using the Display Program (DSPPGM) command.

You can use the QCLRPGMI API to:

- Determine program creation information
- Determine program statistics
- Determine program performance information
- Determine SQL/400 statement information

Authorities and Locks

Library Authority *USE
Program Authority *USE
Program Lock *SHRRD

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. The maximum size for this area is 8 bytes. You can specify the size of this area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If this value is larger than the actual size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The content and format of the information returned for the program. The possible format names are:

PGMI0100 Basic program information.
PGMI0200 Basic program plus SQL/400 statement information.

Program and library name
INPUT; CHAR(20)
The first 10 characters contain the program name, and the second 10 characters contain the name of the library where the program is located. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

PGMI0100 Format

The following information is returned for the PGMI0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 24-23.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
Program creation information			
8	8	CHAR(10)	Program name
18	12	CHAR(10)	Program library name
28	1C	CHAR(10)	Program owner
38	26	CHAR(10)	Program attribute
48	30	CHAR(13)	Creation date and time
61	3D	CHAR(10)	Source file name
71	47	CHAR(10)	Source file library name
81	51	CHAR(10)	Source file member name

Offset		Type	Field
Dec	Hex		
91	5B	CHAR(13)	Source file updated date and time
104	68	CHAR(1)	Observable information
105	69	CHAR(1)	User profile option
106	6A	CHAR(1)	Use adopted authority
107	6B	CHAR(1)	Log commands (CL)
108	6C	CHAR(1)	Allow RTVCLSRC (CL)
109	6D	CHAR(1)	Ignore decimal data
110	6E	CHAR(50)	Text description
160	A0	CHAR(60)	Reserved
Program statistics information			
220	DC	BINARY(4)	Minimum number of parameters
224	E0	BINARY(4)	Maximum number of parameters
228	E4	BINARY(4)	Program size
232	E8	BINARY(4)	Associated space size
236	EC	BINARY(4)	Static storage size
240	F0	BINARY(4)	Automatic storage size
244	F4	BINARY(4)	Number of MI instructions
248	F8	BINARY(4)	Number of MI ODT entries
252	FC	CHAR(1)	Program state
253	FD	CHAR(14)	Compiler identification
267	10B	CHAR(6)	Earliest release that program can run
273	111	CHAR(52)	Reserved
Program performance information			
325	145	CHAR(1)	Optimization
326	146	CHAR(1)	Paging pool
327	147	CHAR(1)	Update PASA
328	148	CHAR(1)	Clear PASA
329	149	CHAR(1)	Paging amount
330	14A	CHAR(18)	Reserved.

PGMI0200 Format

The following information is returned for the PGMI0200 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from the PGMI0100 format.
Program SQL information			
348	15C	BINARY(4)	Number of SQL statements
352	160	CHAR(18)	Relational database
370	172	CHAR(10)	Commitment control
380	17C	CHAR(10)	Allow copy of data
390	186	CHAR(10)	Close SQL cursors
400	190	CHAR(10)	Naming convention
410	19A	CHAR(10)	Date format
420	1A4	CHAR(1)	Date separator
421	1A5	CHAR(10)	Time format
431	1AF	CHAR(1)	Time separator
432	1B0	CHAR(10)	Delay PREPARE
442	1BA	CHAR(10)	Allow blocking

Field Descriptions

For more detailed information than that provided in the following field descriptions, refer to documentation for the create command that was used to create the program. For information on nonSQL fields, this would normally be a create command of the form CRTxxxPGM described in the programmer's guide for the language identified by the xxx value of the command name. For information on SQL fields, this would normally be a command of the form CRTSQLxxx described in the *SQL/400* Programmer's Guide*, where the xxx value of the command name identifies the base language (RPG, COBOL, and so on) of the program the SQL statements are in.

Allow blocking. If blocking will be used to improve the performance of certain SQL statements. The possible values are *NONE, *READ, and *ALLREAD. This information is blank if the program does not contain SQL statements.

Allow copy of data. Whether or not a copy of the data can be used in the implementation of an SQL query. The possible values are *NO, *YES, and *OPTIMIZE. This information is blank if the program does not contain SQL statements.

Allow RTVCLSRC (CL program). The compiler allowed you to control this attribute through the ALWRTVSRC parameter if this program was created using the Create CL Program (CRTCLPGM) command. The possible values are:

N Source for the CL program is not saved with the program (*NO).
Y Source is saved (*YES).

Source that is saved can be retrieved by using the Retrieve CL Source (RTVCLSRC) command. This information is blank if the program is not a CL program.

Associated space size. The size (in bytes) of the associated space used by this program.

Automatic storage size. The size (in bytes) of the automatic storage used by this program.

Bytes available. The length of all data available for the requested format. All available data is returned if enough space is provided.

Bytes returned. The length of the data actually returned.

Clear program automatic storage area (PASA). The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program. The possible values are:

N Do not clear PASA storage (*NOCLRPASA).
C Clear PASA storage (*CLRPASA).

*NOCLRPASA reduces the time needed to call the program, but if a program variable is referred to before it has been set, it may contain unpredictable data.

*CLRPASA increases the time needed to call the program but is used by some programs to ensure that if a program variable is referred to before it has been set, it will contain binary zeros instead of unpredictable data.

Close SQL cursor. When SQL cursors are implicitly closed and SQL prepared statements are implicitly discarded. The possible values are *ENDPGM, *ENDSQL, and *ENDJOB. This information is blank if the program does not contain SQL statements.

Commitment control. The level of commitment control that was specified on the SQL precompile. The possible values are *NONE, *CHG, *CS, and *ALL. This information is blank if the program does not contain SQL statements.

Compiler identification. The licensed program identifier, version number, release level, and modification level of the compiler. The field has a ppppppbVvRrMm format where:

ppppppp The licensed program identifier.
b A blank character.
Vv The character V is followed by a 1-character version number.
Rr The character R is followed by a 1-character release level.
Mm The character M is followed by a 1-character modification level.

For programs created by the Create Program (QPRCRTPG) API, this field identifies the version of the operating system that the program was created under.

The field may be blank if the program is created without going through a compilation process.

Creation date and time. The date and time the program was created. The creation date and time field is in the CYYMMDDHHMMSS format where:

C Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
YY Year
MM Month
DD Day
HH Hour
MM Minute
SS Second

Date format. The format used when accessing date result columns. All output date fields are returned in this format. For input date strings, the value you specify is used to determine whether the date is a valid format. The values returned are *FILE, *USA, *ISO, *EUR, *JIS, *MDY, *DMY, *YMD, *JUL. This information is blank if the program does not contain SQL statements.

Date separator. The separator used when accessing date result columns. This information is blank if the program does not contain SQL statements, however, the number of SQL statements field should be checked to determine if the program contains SQL statements. This is because a blank may be specified as a separator value.

Delay PREPARE. Whether or not PREPARE processing can be delayed until the statement is actually used. The possible values are *YES and *NO. This information is blank if the program does not contain SQL statements.

Earliest release that program can run. The version number, release level, and modification level of the earliest release the program is allowed to run on. The compiler may have allowed you to control this through the TGTRLS parameter of the command used to create the program. The field has a VvRrMm format where:

Vv The character V is followed by a 1-character version number.
Rr The character R is followed by a 1-character release level.
Mm The character M is followed by a 1-character modification level.

Ignore decimal data. The compiler may have allowed you to control this attribute through the ignore decimal data error (IGNDECERR) parameter of the command used to create the program. The possible values are:

N An error is signaled to the program without correcting the data that is not valid (*NO).
Y Decimal data that is not valid is corrected (*YES).

Log commands (CL program). The value specified for the LOG parameter of the CRTCLPGM command. This field is meaningful only if the program is a CL program. command. The possible values are N (*NO), Y (*YES), J (*JOB). This information is blank if the program is not a CL program.

Maximum number of parameters. The maximum number of parameters that may be received by the program when it is called. A value of negative 1 is returned if the program is not observable.

Minimum number of parameters. The minimum number of parameters that is to be received by the program when it is called. A value of negative 1 is returned if the program is not observable.

Naming convention. The convention used for naming objects in SQL statements. The possible values are *SQL and *SYS. This information is blank if the program does not contain SQL statements.

Number of MI instructions. The number of machine interface (MI) instructions used by this program. A value of negative 1 is returned if the program is not observable.

Number of MI ODT entries. The number of ODT (object definition table) entries for the program. This is the number of program objects declared by the compiler. Program objects include variables, constants, labels, operand lists, and exception descriptions. Typically, one or more ODT entries are used for each variable, constant, and label in your program, and some are used by the compiler for internal purposes. The number of internal ODT entries varies depending on the size and complexity of the program. There is a limit of 32,767 ODT entries in a program. A value of negative 1 is returned if the program is not observable.

Number of SQL statements. The number of SQL/400 statements contained in the program. This value is zero if the program does not contain SQL statements.

Observable information. An indicator of whether the observable information associated with the program exists or not. The possible values are:

A The observable information exists (*ALL)
N The observable information does not exist (*NONE).

The observable information for most programs may be removed by using the RMVOBS option on the CHGPGM command.

Optimization. A value of N indicates *NOOPTIMIZE was specified on the OPTIMIZE parameter when the program was created or changed. A value of O indicates *OPTIMIZE was specified.

Paging amount. The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program. The possible values are:

N Page the program one page at a time (*NOBLOCK).
B Page the program in eight-page blocks (*BLOCK).

*BLOCK gives better performance in most situations, because it is likely that more than one page in the block will be referred to before being replaced by some other paging occurring in the storage pool.

*NOBLOCK can give better performance if the other pages that would have been brought in as a block are unlikely to be referred to before being replaced by some other paging occurring in the storage pool.

Paging pool. The paging pool used for the program object. The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program. The values returned are:

U Use the user pool (*USER).
B Use the base pool (*BASE).
M Use the machine pool (*MACHINE).

*USER is used by most system programs and all user programs, unless GENOPT(*MACHINE) is specified.

*BASE is used by certain system programs to avoid disturbing the user pool when they need to be paged in. These programs are not used frequently enough to belong in the machine pool.

*MACHINE is used by a very small number of system programs that are so highly used that their pages should remain almost constantly in main storage. The machine pool is intended to be a stable, low paging pool. If user programs page in the machine pool, there may be contention for main storage between system and user programs, adversely affecting system performance and response times. To prevent paging contention, increase the QMCHPOOL system value with the Change System Value (CHGSYSVAL) command.

Program attribute. The language the program is written in (for example, for a CL program the value is CLP or for an RPG program the value is RPG). This field can be blank (for example, if the program was created by the Create Program (QPRCRTPG) API or if the program is created by a compilation process internal to IBM).

Program library name. The name of the library containing the program.

Program name. The name of the program.

Program owner. The name of the program owner's user profile.

Program size. The size (in bytes) of this program.

Program state. The state of the program. The possible values are:

I The program runs under (inherits) the same state as its caller.
S The program runs as a system state program.
U The program runs as a user state program.

Relational database. The default relational database that was specified on the SQL precompile. *LOCAL indicates that the program can only access data on the local system. A nonblank value other than *LOCAL specifies the name of the relational database to be resolved to through the relational database index. This information is blank if the program does not contain SQL statements.

Reserved. An ignored field.

Source file library name. The name of the library that contains the source file used to create the program . The field is blank if no source file created the program .

Source file member name. The name of the member in the source file. The field is blank if no source file created the program.

Source file name. The name of the source file used to create the program. The field is blank if no source file created the program.

Source file updated date and time. The date and time the member in the source file was last updated. The field is in

the same format as the creation time and date. The field is blank if no source file created the program.

Static storage size. The size (in bytes) of the static storage used by this program.

Text description. The user text, if any, used to briefly describe the program and its function.

Time format. The format used when accessing time result columns. All output time fields are returned in this format. The values returned are *FILE, *USA, *ISO, *EUR, *JIS, *HMS. This information is blank if the program does not contain SQL statements.

Time separator. The separator used when accessing time result columns. This information is blank if the program does not contain SQL statements, however, the number of SQL statements field should be checked to determine if the program contains SQL statements. This is because a blank may be specified as a separator value.

Update program automatic storage area (PASA). The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program. The possible values are:

N Do not update internal PASA stack information (*NOUPDPASA).
U Update internal PASA stack information (*UPDPASA).

*NOUPDPASA reduces the time needed to call the program.

*UPDPASA increases the time needed to call the program but is used by some system programs that are dependent on the updating of the internal PASA stack information.

Use adopted authority. The value specified for the USEADPAUT option on the command used to change the program. The possible values are:

Y Program adopted authority from previous invocation levels is used when this program is running (*YES).
N Program adopted authority from previous invocation levels is NOT used when this program is running (*NO).

User profile option. The value specified for the USRPRF option on the command used to create the program. The possible values are:

U The program runs under the current user's user profile (*USER).
O The program runs under both the current user's and the owner's user profile (*OWNER).

Error Messages

CPF2150 E Object information function failed.
CPF2151 E Operation failed for &2 in &1 type *&3.
CPF24B4 E Severe error occurred while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF8103 E Command &4 in &9 damaged.
CPF8122 E &8 damage on library &4.
CPF8123 E Damage on object information for library &4.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in library &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9806 E Cannot perform function for object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9820 E Not authorized to use library &1.
CPF9821 E Not authorized to program &1 in library &2.
CPF9830 E Cannot assign library &1.

Part 12. Security APIs

Chapter 25. Security APIs

The OS/400 security APIs let you combine many individual jobs into a single server or overhead job without compromising system security. These APIs can be used to consolidate server jobs to reduce processing time and storage use because the system performs job management tasks for only one job. It also speeds response time for system users.

The APIs in this chapter are presented in alphabetical order.

The security APIs and their functions follow:

Change User Password (QSYCHGPW) changes a user's password.
Check User Authority to an Object (QSYCUSRA) returns an indication about a user's specified authority to an object.
Check User Special Authorities (QSYCUSRS) returns an indication of a user's special authorities.
Convert Authority Values to MI Value (QSYCVTA) converts authority values to the machine interface (MI) representation of the value.
Get Profile Handle (QSYGETPH) validates a user ID and password, and creates an encrypted abbreviation called a profile handle for that user profile.
List Authorized Users (QSYLAUTU) puts a list of authorized users of the system in a user space.
List Objects Secured by Authorization List (QSYLATLO) puts a list of objects secured by an authorization list in a user space.
List Objects That Adopt Owner Authority (QSYLOBJP) puts a list of objects that adopt an owner's authority in a user space.
List Objects User Authorized to or Owns (QSYLOBJA) puts a list of objects that a user is authorized to and/or owns in a user space.
List Users Authorized to Object (QSYLUSRA) puts a list of users privately authorized to an object in a user space.
Release Profile Handle (QSYRLSPH) deletes a profile handle.
Retrieve Information about a User (QSYRUSRI) returns the information about a user.
Retrieve User Authority to Object (QSYRUSRA) returns the user's authority to an object.
Set Profile (QWTSETP) switches the job to run under a new profile.

For general information about OS/400 system security, see the *Security Reference* manual.

Change User Password (QSYCHGPW) API

Parameters

Required Parameter Group:

1	User	Input	Char(10)
2	Current password	Input	Char(10)
3	New password	Input	Char(10)
4	Error code	I/O	Char(*)

The Change User Password (QSYCHGPW) API changes a user's password. You must know the existing password that you want to change.

This API provides support similar to the Change Password (CHGPWD) command.

Authorities and Locks

If the user parameter is not *CURRENT or the current user for the job, the caller of the API must have *SECADM special authority and *OBJMGT and *USE authorities to the user profile being changed to change the password.

If the caller of the API:

- Enters the wrong password for the user and,
- Exceeds the maximum number of times allowed by the system value QMAXSIGN, and
- The system value QMAXSGNACN is set to disable user profiles,

then the user profile specified on the user parameter is disabled.

You cannot specify the following user profile names for the user parameter:

```
QDFTOWN   QFNC      QLPINSTALL QSPLJOB
QDOC      QGATE     QSNADS     QTSTRQS
QDSNX     QLPAUTO   QSPL
```

When the new password is checked to ensure it meets the password composition rules for the system, only one error is returned per API call. Therefore, if the new password fails more than one of the rules, multiple calls to the API are needed to determine a correct new password.

You should avoid calling this API from a command line. If this API is called from CL and CL commands are being logged for the job or CL program, the call parameters for the API are logged in the job log. This means the passwords appear in the job log.

Required Parameter Group

User
INPUT; CHAR(10)
The name of the user whose password is being changed. You can specify the following special value:

***CURRENT** The password of the user currently running is changed.

Current password
INPUT; CHAR(10)
The current password for the user. Verification is done to ensure this is the correct password for the user before the password is changed. You can specify the following special value:

***NONE** The user currently does not have a password.

New password
INPUT; CHAR(10)
The new password for the user. Verification is done to ensure the new password meets the password composition rules of the system. You can specify the following special value unless the new password is *NONE:

***NONE** The user is changed to not have a password. This value is not allowed if *CURRENT, the current user name for the job, or QSECOFR is specified on the user parameter.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF0001 E Error found on &1 command.
CPF22C0 E Password does not meet password rules. Return code &1.
CPF22C2 E Password less than &1 characters.
CPF22C3 E Password longer than &1 characters.
CPF22C4 E Password matches one of 32 previous passwords.
CPF22C5 E Password contains one of the following: &1.
CPF22C6 E Password contains two numbers next to each other.
CPF22C7 E Password contains a character used more than once.
CPF22C8 E Same character in same position as previous password.
CPF22C9 E Password must contain a number.
CPF22D1 E Password cannot be same as user ID.
CPF22D2 E Password approval program &1 not found.
CPF22D3 E Password approval program signaled an error.
CPF22D4 E Not allowed to use password approval program.
CPF22D5 E Parameters in password approval program not correct.
CPF22E2 E Password not correct for user profile &1.
CPF22E3 E User profile &1 is disabled.
CPF22F5 E Value &1 for new password not allowed.
CPF22F6 E New password cannot be *NONE.
CPF2203 E User profile &1 not correct.
CPF2225 E Not able to allocate internal system object.
CPF2292 E *SECADM required to create or change user profiles.
CPF2299 E Not able to create internal security object.
CPF3CF1 E Error code parameter not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.

Check User Authority to an Object (QSYCUSRA) API

Parameters

Required Parameter Group:

1	Authority indicator	Output	Char(1)
2	Qualified user name	Input	Char(10)
3	Object name	Input	Char(20)
4	Object type	Input	Char(10)
5	Authority	Input	Char(*)
6	Number of authorities	Input	Binary(4)
7	Call level	Input	Binary(4)
8	Error code	I/O	Char(*)

The Check User Authority to Object (QSYCUSRA) API provides an indication of whether the user has the specified authority to an object.

Authorities and Locks

The following authority is required for the user calling this API, unless the user parameter is *CURRENT or the current user for the job, or the caller owns the object, or the object is an authorization list.

- *OBJMGT authority to the object.
- *READ authority to the user profile.

If the user is *CURRENT, or the current user for the job, the authority to the user includes any authority specified on the object (either private, group, authorization list, or public) plus any program adopted authority. If the user specified is not *CURRENT, the authority available to the user is the authority specified on the object; no program adopted authority is used.

Adopted authority is authority given to the user by the program for the duration of that program. If previous programs in the program stack adopt their owner's authority, the adopted authority for the current program is the accumulated adopted authority from all other programs in the program stack that adopt authority.

When the API checks for authority to logical files, and any data authorities (read, add, update, and delete), are specified, the authority check fails, unless the user adopts authority or has *ALLOBJ special authority. Logical files do not have data authorities associated with them, the data authorities are associated with the base physical file.

Required Parameter Group

Authority indicator
OUTPUT; CHAR(1)
Whether the user has the specified authority to the object. The field contains one of the following:

Y The user has the specified authority.
N The user does not have the specified authority.

Qualified user name
INPUT; CHAR(10)
The name of the user whose authority is checked.

You can specify the following special value:

***CURRENT** Checks the authority of the current user to the specified object.

Object name
INPUT; CHAR(20)
The name of the object whose authority is checked. The first 10 characters specify the object name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the object. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the object.

Object type
INPUT; CHAR(10)
The type of object whose authority is checked.

Authority
INPUT; CHAR(*)
The authority to check for. This parameter can contain up to eight 10-character fields. The following identifies the type of authority the user has to the object:

***EXCLUDE** Exclude authority. If this value is specified, no other values can be specified.
***ALL** All authority.
***CHANGE** Change authority.
***USE** Use authority.
***AUTLMGT** Authorization list management authority. This value is only valid if the object type is *AUTL.
***OBJOPR** Object operational authority.
***OBJMGT** Object management authority.
***OBJEXIST** Object existence authority.
***READ** Read authority.
***ADD** Add authority.
***UPD** Update authority.
***DLT** Delete authority.

Number of authorities
INPUT; BINARY(4)
The number of authorities specified in the authority parameter. You can specify 1 through 8 authorities.

Call level
INPUT; BINARY(4)
The number of call levels to back up in the program stack to do the authority check. For example, if the program that calls this API adopts authority, you would probably not want the authority check to use the adopted authority. Therefore, the authority check should be done at the call level previous to the current level. This parameter should then contain a 1. You can check the authority at the various call levels by signifying a numeric equivalent to the call level. For example, to check the authority at the current call level, specify a 0; to check the authority at the previous call level, specify a 1.

This parameter is only used if the user parameter is *CURRENT or the current user for the job.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF22FA E Authority value &1 not valid.
CPF22FB E Must specify *EXCLUDE or *AUTL as only authority value.
CPF22F7 E Number of authorities must be between 1 and &1.
CPF22F9 E Call level &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3C31 E Object type &1 is not valid.
CPF8122 E &8 damage on library &4.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9812 E File &1 in library &2 not found.
CPF9814 E Device &1 not found.
CPF9820 E Not authorized to use library &1.

Check User Special Authorities (QSYCUSRS) API

Parameters

Required Parameter Group:

1	Authority indicator	Output	Char(1)
2	User name	Input	Char(10)
3	Special authority	Input	Char(*)
4	Number of authorities	Input	Binary(4)
5	Call level	Input	Binary(4)
6	Error code	I/O	Char(*)

The Check User Special Authorities (QSYCUSRS) API provides an indication of whether the user has the specified special authorities.

Authorities and Locks

User Profile Authority
*READ

When the API checks for special authorities and the user parameter is *CURRENT or the current user for the job, the special authorities available to the user include any special authorities the user or the group has, and any program

adopted special authorities. If the user specified is not the user currently running, then the special authorities available to the user are only the special authorities the user and his group have.

If previous programs in the program stack adopt their owner's authority, the adopted authority for the current program is the accumulated adopted authority from all other programs in the program stack that adopt authority.

Required Parameter Group

Authority indicator
OUTPUT; CHAR(1)
Whether the user has the specified special authorities. This parameter contains one of the following:

Y The user has the specified special authorities.
N The user does not have the specified special authorities.

User name
INPUT; CHAR(10)
The name of the user whose special authorities are checked.

You can specify the following special value:

***CURRENT** The special authorities for the user currently running are checked.

Special authority
INPUT; CHAR(*)
The special authorities checked for the user. This parameter can contain up to six 10-character fields. Each of the 10-character fields can contain one of the following special values.

***ALLOBJ** All object special authority.
***SECADM** Security administrator special authority.
***JOBCTL** Job control special authority.
***SPLCTL** Spool control special authority.
***SAVSYS** Save system special authority.
***SERVICE** Service special authority.

Number of authorities
INPUT; BINARY(4)
The number of special authorities specified in the special authority parameter. You can specify 1 through 6.

Call level
INPUT; BINARY(4)
The number of call levels to back up in the program stack to do the authority check. For example, if the program that calls this API adopts authority, you would probably not want the authority check to use the adopted authority. Therefore, the authority check should be done at the call level previous to the current level. This parameter should then contain a 1. You can check the authority at the various call levels by signifying a numeric equivalent to the call level. For example, to check the authority at the current call level, specify a 0; to check the authority at the previous call level, specify a 1.

This parameter is only used if the user parameter is *CURRENT, or the current user name for the job.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF22F7 E Number of authorities must be between 1 and &1.
CPF22F8 E Special authority value &1 not valid.
CPF22F9 E Call level &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF8122 E &8 damage on library &4.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.

Convert Authority Values to MI Value (QSYCVTA) API

Parameters

Required Parameter Group:

1	Converted authority value	Output	Char(2)
2	Authority special value	Input	Char(*)
3	Number of authorities	Input	Binary(4)
4	Error code	I/O	Char(*)

The Convert Authority Values to MI Value (QSYCVTA) API converts the special values indicating authority to the corresponding machine interface (MI) representation of that value.

Required Parameter Group

Converted authority value
OUTPUT; CHAR(2)
The MI representation of the authority special value in hexadecimal format.

Authority special value
INPUT; CHAR(*)
The authority special values that are converted. The converted value is the cumulative value of all authority special values specified. This parameter can contain up to eight 10-character fields. Each of the 10-character fields can contain one of the following special values. The following identifies the authority special values that are converted to the corresponding MI representation of that value.

***AUTL**	Authorization list authority. If this value is specified, no other values can specified. This authority value is only valid for *PUBLIC authority on an object secured by an authorization list.
***EXCLUDE**	Exclude authority. If this value is specified, no other values can be specified.
***ALL**	All authority.
***CHANGE**	Change authority.
***USE**	Use authority.
***OBJOPR**	Object operational authority.
***OBJMGT**	Object management authority.
***OBJEXIST**	Object existence authority.
***READ**	Read authority.
***ADD**	Add authority.
***UPD**	Update authority.
***DLT**	Delete authority.
***AUTLMGT**	Authorization list management authority.

Number of authorities
INPUT; BINARY(4)
The number of authority special values specified in the authority special value parameter. You can specify 1 through 8.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF22FA E Authority value &1 not valid.
CPF22FB E Must specify *EXCLUDE or *AUTL as only authority value.
CPF22F7 E Number of authorities must be between 1 and &1.
CPF3CF1 E Error code parameter not valid.

Get Profile Handle (QSYGETPH) API

Parameters

Required Parameter Group:

1	User ID	Input	Char(10)
2	Password	Input	Char(10)
3	Profile handle	Output	Char(12)

Optional Parameter:

4	Error code	I/O	Char(*)

The Get Profile Handle (QSYGETPH) API validates user IDs and passwords and creates a profile handle, a type of encrypted pointer, for use in jobs that run under more than one user profile. The profile handle is temporary; you can use it only in the job that created it. In addition, you should use the QSYGETPH API and the Set Profile (QWTSETP) API only to change profiles in server-type jobs. To verify a user's password for other purposes, use the Check Password (CHKPWD) command.

The QSYGETPH API follows this process:

- Verifies that the user ID and password are correct. Incorrect passwords and special cases are handled as follows:
  - On level 10 systems, only the user ID is validated because no passwords are required.
  - If the password is not correct, the incorrect password count is increased. (The QMAXSIGN system value contains the maximum number of incorrect attempts to sign on.) If the QMAXSGNACN system value is set to disable the user profile, repeated attempts to validate an incorrect password disable the user ID. This keeps applications from methodically determining user passwords.
  - If the user ID is *CURRENT, the QSYGETPH API does not verify the password.
  - If the password is *NOPWD, the user requesting the profile handle must have *ALLOBJ and *SECADM special authorities.
- If the user profile is disabled, the password is expired, or the user's password is *NONE, ends without generating a profile handle.
- Generates the profile handle, a 12-character random string designating the user's authorities. This string, not the user's password, supplies the Set Profile (QWTSETP) and the Release Profile Handle (QSYRLSPH) APIs. A single job can create up to 65 534 profile handles; after that, the space to store them is full. Message CPF22E6 is sent to the application, and QSYGETPH stops generating profile handles.

  Be sure to keep track of the profile handles created in the calling application. If the application calls QSYGETPH twice with the same user profile and password, QSYGETPH returns two different profile handles. Either handle can be used, but generating and using just one is more efficient.
- Updates the last-used date for the user and group profiles.
- Resets the signon attempts not valid count to zero.
- If security-related events are being audited, adds an entry to the QAUDJRN audit journal to indicate that a profile handle is created.

Authorities and Locks

If the password is *NOPWD, *ALLOBJ and *SECADM, special authorities are required for the user requesting the profile handle.

Required Parameter Group

User ID
INPUT; CHAR(10)
The user ID of the profile for which the handle is being created.

You can specify the following special value:

***CURRENT** The handle for the current user profile is generated without validating the password.

Password
INPUT; CHAR(10)
The password for the user ID. On security level 10 systems, the password is not required.

You can specify the following special value:

***NOPWD** If the user currently running has *ALLOBJ and *SECADM special authorities, no passwords are validated.

The QSYGETPH API does not create a handle for a disabled user profile, one with a password of *NONE, or one with an expired password.

Profile handle
OUTPUT; CHAR(12)
A unique string or handle designating the user profile to use as input to other routines. The handle is temporary; you can use it only in the job that created it.

No profile handle is created if the user profile is disabled, the password has expired, or the password is *NONE.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF22E2 E Password not correct for user profile &1.
CPF22E3 E User profile &1 is disabled.
CPF22E4 E Password for user profile &1 has expired.
CPF22E5 E No password associated with user profile &1.
CPF22E6 E Maximum number of ProfileHandles have been generated.
CPF22E9 E *ALLOBJ and *SECADM special authority needed.
CPF2203 E User profile &1 not correct.
CPF2204 E User profile &1 not found.
CPF2213 E Not able to allocate user profile &1.
CPF2225 E Not able to allocate internal system object.
CPF2299 E Not able to create internal security object.

List Authorized Users (QSYLAUTU) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Error code	I/O	Char(*)

The List Authorized Users (QSYLAUTU) API puts a list of authorized system users into a user space.

This API provides information similar to the Display Authorized Users (DSPAUTUSR) command.

Authorities and Locks

User Space Authority
*CHANGE
Authority to Library Containing User Space
*USE
Authority to User Profiles in List of Authorized Users
*READ, only those profiles that you have *READ authority to are returned.

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space that the list of authorized users is returned to. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is searched for the user space. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is searched for the user space.

Format name
INPUT; CHAR(8)
The name of the format used to list the authorized users.

You can specify these formats:

AUTU0100 Each entry contains the user name and group name. For a detailed description of this format, see "AUTU0100 Format" on page 25-7.
AUTU0200 Each entry contains the same information as AUTU0100 plus the text description for the user. For a detailed description of this format, see "AUTU0200 Format" on page 25-7.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions."

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name

AUTU0100 Format

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User name
10	0A	CHAR(10)	Group name

AUTU0200 Format

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User name
10	0A	CHAR(10)	Group name
20	14	CHAR(50)	Text name

Field Descriptions

Format name. The name of the format used to list authorized users.

Group name. The name of the user's group profile. If the user does not have a group profile, this field contains *NONE.

User space library name. The name of the library containing the user space.

Text name. The text description for the authorized user.

User name. The name of the authorized user.

User space name. The name of the user space used to return the list of authorized users on the system.

Error Messages

CPF2225 E Not able to allocate internal system object.
CPF2299 E Not able to create internal security object.
CPF3CAA E List is too large for user space &1.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.

List Objects Secured by Authorization List (QSYLATLO) API

(QSYLATLO) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Authorization list	Input	Char(10)
4	Error code	I/O	Char(*)

The List Objects Secured by Authorization List (QSYLATLO) API puts a list of objects secured by an authorization list into a user space.

This API provides information similar to the Display Authorization List Objects (DSPAUTLOBJ) command.

Authorities and Locks

User Space Authority
*CHANGE
Authority to Library Containing User Space
*USE
Authorization List Authority
Must not be *EXCLUDE authority

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space where the list of objects secured by the authorization list is returned to. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the user space. If there is no current library, QGPL (general purpose library)
***LIBL** The library list is used to locate the user space.

Format name
INPUT; CHAR(8)
The name of the format used to list objects secured by the authorization list.

You can specify these formats:

ATLO0100 Each entry contains the object name, library, type, and authority holder indicator. For a detailed description of this format, see "ATLO0100 Format" on page 25-8.

ATLO0200 Each entry contains the same information as ATLO0100 plus the object owner, attribute, and text. For a detailed description of this format, see "ATLO0200 Format" on page 25-8.

Authorization list
INPUT; CHAR(10)
The name of the authorization list for which the secured objects are returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions."

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	Authorization list

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Authorization list
10	0A	CHAR(10)	User space library name
20	14	CHAR(10)	Owner

ATLO0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder

ATLO0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder
31	1F	CHAR(10)	Owner
41	29	CHAR(10)	Attribute
51	33	CHAR(50)	Text description

Field Descriptions

Attribute. The attribute of the secured object.

Authority holder. Whether the object is an authority holder. If the object is an authority holder, this field is Y. If not, this field is N.

Authorization list. The name of the authorization list for which the list of objects is returned.

Format name. The name of the format used to list objects secured by the authorization list.

Library name. The name of the library containing the user space, object, or authorization list.

Object name. The name of the object secured by the authorization list.

Object type. The type of secured object.

Owner. The name of the owner of the authorization list or object.

Text description. The descriptive text for the secured object.

User space. The user space used to return the list of objects secured by the authorization list.

Error Messages

CPF22AF E Not authorized to authorization list &1.
CPF2283 E Authorization list &1 does not exist.
CPF2289 E Unable to allocate authorization list &1.
CPF3CAA E List is too large for user space &1.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.

List Objects That Adopt Owner Authority (QSYLOBJP) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	User	Input	Char(10)
		name	
4	Object type	Input	Char(10)
5	Continuation handle	Input	Char(20)
6	Error code	I/O	Char(*)

The List Objects That Adopt Owner Authority (QSYLOBJP) API puts a list of objects that adopt an object owner's authority into a user space.

This API provides information similar to that provided by the Display Program Adopt (DSPPGMADP) command.

Authorities and Locks

User Space Authority *CHANGE
Authority to Library Containing User Space *USE
User Profile Authority *OBJMGT

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space to which the list of objects that adopt a user's authority is returned. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the user space. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the user space.

Format name
INPUT; CHAR(8)
The name of the format that returns information on the objects that adopt a user's authority.

You can specify these formats:

OBJP0100 Each entry contains the object name, library, type, and object in use indicator. For a detailed description of this format, see "OBJP0100 Format" on page 25-10.
OBJP0200 Each entry contains the same information as format OBJP0100 plus the object attribute and descriptive text. For a detailed description of this format, see "OBJP0200 Format" on page 25-10.

User name
INPUT; CHAR(10)
The user name for which the list of objects that adopt the user's authority is returned. You can specify the following special value:

***CURRENT** The list of objects that adopt the authority of the user currently running is returned. If *CURRENT is used, the name of the current user is returned in the list header section of the user space.

Object type
INPUT; CHAR(10)
The type of object for which the list of objects that adopt the user's authority is returned. You can specify only the following special values:

***ALL** Return entries for all object types that adopt authority.
***PGM** Return entries for programs that adopt authority.
***SQLPKG** Return entries for SQL packages that adopt authority.

Continuation handle
INPUT; CHAR(20)
The handle used to continue from a previous call to this API that resulted in partially complete information. You can determine if a previous call resulted in partially complete information by checking the Information Status variable in the generic user space header following the API call.

If the API is not attempting to continue from a previous call, this parameter must be set to blanks. Otherwise, a valid continuation value must be supplied. The value may be obtained from the list header section of the user space used in the previous call. When continuing, the first entry in the returned list is the entry that immediately follows the last entry returned in the previous call.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions" on page 25-10.

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	User space library name

Offset		Type	Field
Dec	Hex		
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	User name
38	26	CHAR(10)	Object type
48	30	CHAR(20)	Continuation handle

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User name
10	0A	CHAR(20)	Continuation handle

OBJP0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
21	15	CHAR(1)	Object in use

OBJP0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
21	15	CHAR(1)	Object in use
31	1F	CHAR(10)	Attribute
41	29	CHAR(50)	Text description

Field Descriptions

Attribute. The object attribute.

Continuation handle (header section). A continuation point for the API.: This value is set based on the contents of the Information Status variable in the generic header for the user space. The following situations can occur:

- Information status - C. The information returned in the user space is valid and complete. No continuation is necessary and the continuation handle is set to blanks.
- Information status - P. The information returned in the user space is valid but incomplete. The user may call the API again, starting where the last call left off. The continuation handle contains a value which may be supplied as an input parameter in later calls.
- Information status - I. The information returned in the user space is not valid and incomplete. The content of the continuation handle is unpredictable.

Continuation handle (input section). Used to continue from a previous call to this API which resulted in partially complete information.

Format name. The name of the format used to return information on the objects that adopt authority.

Library name. The name of the library containing the user space or object.

Object name. The name of the object that adopts the user's authority.

Object in use. Whether the object is in use when the API tries to access it. If the object is in use, the API is not able to determine if the object adopts the user's authority. If the object is in use, this field is Y. If not, this field is N.

Object type.

- Input Section: The type of object for which the list of objects adopting the user's authority is returned.
- List Section: The type of object which adopts the user's authority.

Text description. The text description of the object.

User name. The name of the user for which the list of objects that adopt the user's authority is returned.

User space name. The name of the user space to which the list of objects that adopt the users authority is returned.

Error Messages

CPF22FD E Continuation handle not valid for API &1.
CPF2204 E User profile &1 not found.
CPF2213 E Not able to allocate user profile &1.
CPF2217 E Not authorized to user profile &1.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C31 E Object type &1 is not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.

List Objects User Is Authorized To or Owns (QSYLOBJA) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	User name	Input	Char(10)
4	Object type	Input	Char(10)
5	Returned objects	Input	Char(10)
6	Continuation handle	Input	Char(20)
7	Error code	I/O	Char(*)

The List Objects a User is Authorized To or Owns (QSYLOBJA) API puts a list of objects a user is authorized to or owns into a user space. The list of authorized objects only includes objects the user is specifically authorized to. The list does not include objects the user is authorized to because:

- The user is part of a group that is authorized
- The user can access the object using the public authority
- The object is secured with an authorization list the user is authorized to
- The user can access the object using adopted authority

This API provides information similar to that provided by the Display User Profile (DSPUSRPRF) command when specifying *OBJAUT or *OBJOWN for the type parameter.

Authorities and Locks

User Space Authority *CHANGE
Authority to Library Containing User Space *USE
User Profile Authority *READ

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space used to return the list of objects a user is authorized to or owns. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the user space. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the user space.

Format name
INPUT; CHAR(8)
The name of the format used to list objects the owner is authorized to and/or owns.

You can specify these formats:

OBJA0100 Each entry contains the object name, library, type, authority holder indicator and ownership indicator. For a detailed description of this format, see "OBJA0100 Format" on page 25-12.
OBJA0200 Each entry contains the same information as format OBJA0100 plus the authority values. For a detailed description of this format, see "OBJA0200 Format" on page 25-12.
OBJA0300 Each entry contains the same information as format OBJA0200 plus the object attribute and descriptive text. For a detailed description of this format, see "OBJA0300 Format" on page 25-12.

User name
INPUT; CHAR(10)
The user name for which the list of authorized or owned objects is being returned. You can specify the following special value:

***CURRENT** The list of objects the user currently signed on is authorized to or owns is returned. If *CURRENT is used, the name of the current user is returned in the list header section of the user space.

Object type
INPUT; CHAR(10)
The type of object the list of authorized or owned objects is returned for. You can specify the following special value:

***ALL** Return entries of all object types.

Returned objects
INPUT; CHAR(10)
The objects that are returned. You can specify the following special values:

***OBJAUT** The list of objects the user is authorized to is returned.
***OBJOWN** The list of objects the user owns is returned.
***BOTH** The list of objects the user is authorized to and owns is returned. The list of owned objects precedes the list of authorized objects.

Continuation handle
INPUT; CHAR(20)
The handle used to continue from a previous call to this API that resulted in partially complete information. You can determine if a previous call resulted in partially complete information by checking the Information Status variable in the generic user space header following the API call.

If the API is not attempting to continue from a previous call, this parameter must be set to blanks. Otherwise, a valid continuation value must be supplied. The value may be obtained from the list header section of the user space used in the previous call.

When continuing, the first entry in the returned list is the entry that immediately follows the last entry returned in the previous call.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions."

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	Library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	User name
38	26	CHAR(10)	Object type
48	30	CHAR(10)	Returned objects
58	3A	CHAR(20)	Continuation handle

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User name
10	0A	CHAR(20)	Continuation handle

OBJA0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder
31	1F	CHAR(1)	Ownership

OBJA0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder
31	1F	CHAR(1)	Ownership
32	20	CHAR(10)	Authority value
42	2A	CHAR(1)	Authorization list management
43	2B	CHAR(1)	Object operational
44	2C	CHAR(1)	Object management
45	2D	CHAR(1)	Object existence
46	2E	CHAR(1)	Data read
47	2F	CHAR(1)	Data add
48	30	CHAR(1)	Data update
49	31	CHAR(1)	Data delete

OBJA0300 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder
31	1F	CHAR(1)	Ownership
32	20	CHAR(10)	Authority value
42	2A	CHAR(1)	Authorization list management
43	2B	CHAR(1)	Object operational
44	2C	CHAR(1)	Object management
45	2D	CHAR(1)	Object existence
46	2E	CHAR(1)	Data read
47	2F	CHAR(1)	Data add
48	30	CHAR(1)	Data update
49	31	CHAR(1)	Data delete
50	32	CHAR(10)	Attribute
60	3C	CHAR(50)	Text description

Field Descriptions

Attribute. The object's attribute.

Authority holder. Whether the object is an authority holder. If the object is an authority holder, this field is Y. If not, this field is N.

Authority value. The special value indicating the user's authority to the object. This field contains one of the following values:

**ALL* The user has all object (operational, management, and existence) and data (read, add, update, and delete) authorities to the object.

| **CHANGE* The user has object operational and all data authorities to the object.
| **USE* The user has object operational and data read authorities to the object.
| **EXCLUDE* The user has none of the object or data authorities to the object, or authorization list management authority.
| *USER DEF* The user has some combination of object and data authorities that do not relate to a special value. The individual authorities for the user should be checked to determine what authority the user has to the object. This value is returned if the user owns an object and all authority for the user to the object has been removed. If this happens, all individual authority fields are set to N.

| **Authorization list management**. Whether the user has authorization list management authority to the object. If the user has the authority, this field is Y. If not, this field is N. This field is only valid if the object type is *AUTL.

| **Continuation handle (header section)**. A continuation point for the API.

| This value is set based on the contents of the Information Status variable in the generic header for the user space. The following situations can occur:

- Information status – C. The information returned in the user space is valid and complete. No continuation is necessary and the continuation handle is set to blanks.
- Information status – P. The information returned in the user space is valid but incomplete. The user may call the API again, picking up where the last call ended. The continuation handle contains a value, that may be supplied as an input parameter in later calls.
- Information status – I. The information returned in the user space is not valid or complete. The contents of the continuation handle are unpredictable.

| **Continuation handle (input section)**. The handle used to continue from a previous call to this API that resulted in partially complete information.

| **Data add**. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

| **Data delete**. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

| **Data read**. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

| **Data update**. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

| **Format name**. The name of the format used to list objects the user is authorized to or owns.

| **Library name**. The name of the library containing the user space or object.

| **Object name**. The name of the object the user is authorized to or owns.

| **Object existence**. Whether the user has object existence authority to the object. If the user has the authority, this field is Y. If not, this field is N.

| **Object management**. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

| **Object operational**. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

| **Object type**.

| *Input Section*
| The type of object for which the list of authorized or owned objects is returned.
| *List Section*
| The type of object the user is authorized to or owns.

| **Ownership**. Whether the user owns the object. If the user owns the object, this field is Y. If not, this field is N.

| **Returned objects**. The objects that are returned.

| **Text description**. The text description of the object.

| **User name**. The user name for which the list of authorized or owned objects is returned.

| **User space name**. The name of the user space used to return the list of objects the user is authorized to or owns.

Error Messages

| CPF22FC E Value &1 not valid when specifying objects to be returned by API &2.
| CPF22FD E Continuation handle not valid for API &1.
| CPF2204 E User profile &1 not found.
| CPF2213 E Not able to allocate user profile &1.
| CPF2217 E Not authorized to user profile &1.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF3C31 E Object type &1 is not valid.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.

List Users Authorized to Object (QSYLUSRA) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified object name	Input	Char(20)
4	Object type	Input	Char(10)
5	Error code	I/O	Char(*)

The List Users Authorized to Object (QSYLUSRA) API puts a list of users privately authorized to an object, including an authorization list, into a user space. The information returned is the authority as it exists for the object. Any authority the process has to the object through its group or adopted authority is not included. *PUBLIC authority to the object is also returned in the first list entry of the user space.

This API provides information similar to that provided by the Display Authorization List (DSPAUTL) command or the Display Object Authority (DSPOBJAUT) command.

Authorities and Locks

User Space Authority *CHANGE
Authority to Library Containing User Space
*USE
Specified Object or Authorization List Authority
*OBJMGT

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space used to return the list of authorized users to the object. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the user space. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the user space.

Format name
INPUT; CHAR(8)
The name of the format used to list authorized users.

You can specify this format:

USRA0100 Each entry contains the user name and authority values. For a detailed description of this format, see "USRA0100 Format."

Qualified object name
INPUT; CHAR(20)
The name of the object for which the list of authorized users is returned. The first 10 characters specify the object name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the object. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the object.

Object type
INPUT; CHAR(10)
The type of object for which the list of authorized users is returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions" on page 25-15.

Input Parameter Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	Library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	Object name
38	26	CHAR(10)	Library name
48	30	CHAR(10)	Object type

Header Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(10)	Owner name
40	28	CHAR(10)	Authorization list

USRA0100 Format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User name
10	0A	CHAR(10)	Authority value
20	14	CHAR(1)	Authorization list management
21	15	CHAR(1)	Object operational

Offset		Type	Field
Dec	Hex		
22	16	CHAR(1)	Object management
23	17	CHAR(1)	Object existence
24	18	CHAR(1)	Data read
25	19	CHAR(1)	Data add
26	1A	CHAR(1)	Data update
27	1B	CHAR(1)	Data delete

Field Descriptions

Authority value. The user's authority to the object. This field contains one of the following values:

**ALL* The user has all object (operational, management, and existence) and data (read, add, update, and delete) authorities to the object.

**CHANGE* The user has object operational and all data authorities to the object.

**USE* The user has object operational and data read authorities to the object.

**EXCLUDE* The user has none of the object or data authorities to the object, or authorization list management authority to the authorization list.

**AUTL* The public authority for the object comes from the public authority on the authorization list securing the object. This value can only be returned if there is an authorization list securing the object and the authorized user is *PUBLIC.

USER DEF The user has some combination of object and data authorities that do not relate to a special value. The individual authorities for the user should be checked to determine what authority the user has to the object.

Authorization list. The name of the authorization list securing the object. If there is no authorization list securing the object, this field is *NONE.

Authorization list management. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N. This field is only valid if the object type is *AUTL.

Data add. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data delete. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data read. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data update. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Format name. The name of the format used to list users authorized to the object.

User space library name. The name of the library containing the user space or object.

Object name. The name of the object for which the list of authorized users is returned.

Object existence. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object management. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object operational. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object type. The type of object for which the list of authorized users is returned.

Owner. The name of the owner of the object. If all authority for the owner is removed, no list entry is returned for the owner.

User name. The name of the user authorized to the object. This field can contain the following special value:

**PUBLIC* Public authority (authority used by users not privately authorized) to the object. This is the first entry in the list data section.

User space name. The name of the user space used to return the list of users authorized to the object.

Error Messages

CPF3CAA E List is too large for user space &1.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C31 E Object type &1 is not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.

Release Profile Handle (QSYRLSPH) API

Parameters

Required Parameter:

1	Profile handle	Input	Char(12)

Optional Parameter:

2	Error code	I/O	Char(*)

The Release Profile Handle (QSYRLSPH) API validates a given profile handle and then deletes it. To use the user profile represented by the deleted profile handle in the job again, you must call the Get Profile Handle (QSYGETPH) API to generate a new profile handle for the user profile.

Your application must perform any needed cleanup work like closing files and deallocating objects. The QSYRLSPH API only releases the profile handle; it does not perform any cleanup in the job.

Required Parameter

Profile handle
INPUT; CHAR(12)
The profile handle to be released.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF3CF1 E Invalid error code structure specified.

Retrieve Information about User (QSYRUSRI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Format name	Input	Char(8)
4	User name	Input	Char(10)
5	Error code	I/O	Char(*)

The Retrieve Information about User (QSYRUSRI) API provides information about a user profile. This API provides information similar to the Retrieve User Profile (RTVUSRPRF) command or the Display User Profile (DSPUSRPRF) command when *BASIC is specified for the type parameter.

Authorities and Locks

User Profile Authority *READ

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable used to return the information about the user. This variable must be at least 8 bytes long.

Receiver variable length
INPUT; BINARY(4)
The length of the receiver variable. The variable must be at least 8 bytes long.

Format name
INPUT; CHAR(8)
The name of the format used to return information about the user.

You can specify these formats:

USRI0100 Sign-on and password information is returned. For a detailed description of this format, see "USRI0100 Format."

USRI0200 Authority information is returned. For a detailed description of this format, see "USRI0200 Format" on page 25-17.

USRI0300 All user information is returned. For a detailed description of this format, see "USRI0300 Format" on page 25-17.

User name
INPUT; CHAR(10)
The user name for which information is returned. You can specify the following special value:

***CURRENT** The information for the user currently running is returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Receiver Variable Description

The following tables describe the order and format of the data returned in the receiver variable. For detailed descriptions of the fields in the tables, see "Field Descriptions" on page 25-18.

USRI0100 Format

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	User profile name
18	12	CHAR(13)	Previous sign-on date and time
31	1F	CHAR(1)	Reserved

Offset		Type	Field
Dec	Hex		
32	20	BINARY(4)	Sign-on attempts not valid
36	24	CHAR(10)	Status
46	2E	CHAR(8)	Password change date
54	36	CHAR(1)	No password indicator
55	37	CHAR(1)	Reserved
56	38	BINARY(4)	Password expiration interval
60	3C	CHAR(8)	Date password expires
68	44	BINARY(4)	Days until password expires
72	48	CHAR(1)	Set password to expire
73	49	CHAR(10)	Display sign-on information

USRI0200 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	User profile name
18	12	CHAR(10)	User class name
28	1C	CHAR(15)	Special authorities
43	2B	CHAR(10)	Group profile name
53	35	CHAR(10)	Owner
63	3F	CHAR(10)	Group authority name
73	49	CHAR(10)	Limit capabilities

USRI0300 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	User profile name
18	12	CHAR(13)	Previous sign-on date and time
31	1F	CHAR(1)	Reserved
32	20	BINARY(4)	Sign-on attempts not valid
36	24	CHAR(10)	Status
46	2E	CHAR(8)	Password change date
54	36	CHAR(1)	No password indicator
55	37	CHAR(1)	Reserved
56	38	BINARY(4)	Password expiration interval
60	3C	CHAR(8)	Date password expires
68	44	BINARY(4)	Days until password expires
72	48	CHAR(1)	Set password to expire
73	49	CHAR(10)	User class name
83	53	CHAR(15)	Special authorities
98	62	CHAR(10)	Group profile name
108	6C	CHAR(10)	Owner
118	76	CHAR(10)	Group authority name
128	80	CHAR(10)	Assistance level
138	8A	CHAR(10)	Current library name
148	94	CHAR(10)	Initial menu name
158	9E	CHAR(10)	Initial menu library name
168	A8	CHAR(20)	Initial program name
178	B2	CHAR(10)	Initial program library name
188	BC	CHAR(10)	Limit capabilities
198	C6	CHAR(50)	Text description
248	F8	CHAR(10)	Display sign-on information
258	102	CHAR(10)	Limit device sessions
268	10C	CHAR(10)	Keyboard buffering
278	116	CHAR(2)	Reserved
280	118	BINARY(4)	Maximum allowed storage
284	11C	BINARY(4)	Storage used
288	120	CHAR(1)	Highest scheduling priority
289	121	CHAR(10)	Job description name
299	126	CHAR(10)	Job description library name
309	135	CHAR(15)	Accounting code
324	144	CHAR(10)	Message queue name
334	14E	CHAR(10)	Message queue library name
344	158	CHAR(10)	Message queue delivery method
354	162	CHAR(2)	Reserved
356	164	BINARY(4)	Message queue severity
360	168	CHAR(10)	Output queue name
370	172	CHAR(10)	Output queue library name
380	17C	CHAR(10)	Print device
390	186	CHAR(10)	Special environment
400	190	CHAR(10)	Attention-key-handling program name
410	19A	CHAR(10)	Attention-key-handling program library name
420	1A4	CHAR(10)	Language ID
430	1AE	CHAR(10)	Country ID
440	1B8	BINARY(4)	Character code set ID

Offset		Type	Field
Dec	Hex		
444	1BC	CHAR(36)	User options

Field Descriptions

Accounting code. The accounting code associated with this user. If the user does not have an accounting code, this field is blank.

Assistance level. The user interface the user will use. The field contains one of the following values:

**SYSVAL* The system value QASTLVL determines which user interface the user is using.
**BASIC* The user uses the Operational Assist user interface.
**INTERMED* The user uses the system user interface.
**ADVANCED* The user uses the expert system user interface.

Attention-key-handling program name. The Attention-key-handling program for this user. This field contains the following fields:

- CHAR(10): The name of the Attention-key-handling program. This field may contain one of the following special values:
  - **SYSVAL* The system value, QATNPGM, is used to determine the user's Attention-key-handling program.
  - **NONE* No Attention-key-handling program is used by this user.
  - **ASSIST* The Operational Assistant Attention-Key-Handling API (QEZMAIN) is used by this user.
- Attention-key-handling program library name: The name of the library where the program is located. This field can contain the special value of *LIBL. If the program name is a special value, this field is blank.

Bytes available. The number of bytes of data available to be returned to the user. If all data is returned, this is the same as the number of bytes returned. If the receiver variable was not big enough to contain all of the data, this is the number of bytes that can be returned.

Bytes returned. The number of bytes of data returned to the user. This is the lesser of the number of bytes available to be returned or the length of the receiver variable.

Character code set ID. The character code set ID to be used by the system for this user. This field can contain the following special values:

-2 The system value QCCSID is used to determine the user's character code set ID.

Country ID. The country ID used by the system for this user. This field can contain the following special value:

**SYSVAL* The system value QCNTRYID is used to determine the user's country ID.

Current library name. This field contains the name of the user's current library. If the user does not have a current library, this field is *CRTDFT.

Date password expires. The date the user's password expires, in *DTS (date-time stamp) format. If the user's password will not expire (password expiration interval of *NOMAX) or the user's password is set to expire, then this field is blank.

Days until password expires. The number of days until the password will expire. This field contains one of the following values:

0 The password is expired.
1 – 7 The number of days until the password expires.
-1 The password will not expire in the next 7 days.

Display sign-on information. Whether the sign-on information display is shown when the user signs on. The field contains one of the following values:

**SYSVAL* The system value QDSPSGNINF determines if the sign-on information display is shown when the user signs on.
**YES* The sign-on information display is shown when the user signs on.
**NO* The sign-on information display is not shown when the user signs on.

Group authority name. The authority the user's group profile has to objects the user creates. The field contains one of the following values:

**NONE* The group profile has no authority to the objects the user creates. If the user does not have a group profile, the field contains this value.
**ALL* The group profile has all authority to the objects the user creates.
**CHANGE* The group profile has change authority to the objects the user creates.
**USE* The group profile has use authority to the objects the user creates.
**EXCLUDE* The group profile has exclude authority to the objects the user creates.

Group profile name. The name of the group profile. If this user does not have a group profile, this field is *NONE.

Highest scheduling priority. The highest scheduling priority the user is allowed to have for each job submitted to the system. The priority is a value from 0 through 9, with 0 being the highest priority.

Initial menu name. The initial menu for the user. This field contains the following fields:

- CHAR(10): The name of the menu. This field can contain the special value *SIGNOFF.
- CHAR(10): The name of the library where the menu is located. This field can contain the special value of *LIBL. If the menu name is *SIGNOFF, this field is blank.

Initial menu library name. The name of the library that the initial menu is in.

Initial program name. The initial program for the user. This field contains the following fields:

- CHAR(10): The name of the program. If the user does not have an initial program, this field is *NONE.
- CHAR(10): The name of the library where the program is located. This field can contain the special value of *LIBL. If the program name is *NONE, this field is blank.

Initial program library name. The name of the library that the initial program is in.

Job description name. The name of the job description used for jobs that start through subsystem work station entries. This field contains the following fields:

- CHAR(10): The name of the job description.
- CHAR(10): The name of the library where the job description is located or the special value *LIBL.

Job description library name. The name of the library that the job description is in.

Keyboard buffering. This field indicates the keyboard buffering value that is used when a job is initialized for this user. The field contains one of the following values:

*SYSVAL	The system value QKBDBUF determines the keyboard buffering value for this user.
*YES	The type-ahead and attention-key buffering options are both on.
*NO	The type-ahead and attention-key buffering options are not on.
*TYPEAHEAD	The type-ahead option is on, but the attention-key buffering option is not.

Language ID. The language ID used by the system for this user. This field can contain the following special value:

*SYSVAL	The system value QLANGID is used to determine the user's language ID.

Limit capabilities. Whether the user has limited capabilities. The field contains one of the following values:

*PARTIAL	The user cannot change his initial program or current library.
*YES	The user cannot change his initial menu, initial program, or current library. The user cannot run commands from the command line.
*NO	The user is not limited.

Limit device sessions. Whether the user is limited to one device session. The field contains one of the following values:

*SYSVAL	The system value QLMTDEVSSN determines if the user is limited to one device session.
*YES	The user is limited to one device session.
*NO	The user is not limited to one device session.

Maximum allowed storage. The maximum amount of auxiliary storage (in kilobytes) that can be assigned to store permanent objects owned by the user. If the user does not have a maximum amount of allowed storage, this field contains -1 for *NOMAX.

Message queue name. The name of the message queue that is used by this user. This field contains the following fields:

- CHAR(10): The name of the message queue.
- CHAR(10): The name of the library where the message queue is located or the special value *LIBL.

Message queue library name. The name of the library the message queue is in.

Message queue delivery method. How the messages are delivered to the message queue used by the user. This field contains one of the following special values:

*BREAK	The job to which the message queue is assigned is interrupted when a message arrives on the message queue.
*DFT	Messages requiring replies are answered with their default reply.
*HOLD	The messages are held in the message queue until they are requested by the user or program.
*NOTIFY	The job to which the message queue is assigned is notified when a message arrives on the message queue.

Message queue severity. The lowest severity that a message can have and still be delivered to a user in break or notify mode. The severity is a value from 0 through 99.

No password indicator. If *NONE is specified for the password in the user profile, this field contains a Y. If not, this field is N.

Output queue name. The output queue used by this user. This field contains the following fields:

- CHAR(10): The name of the output queue used by the user. This field can contain one of the following special values:
  - *WRKSTN The output queue assigned to the user's work station is used.
  - *DEV An output queue with the same name as the device specified in the printer device parameter is used by the user.
- CHAR(10): The name of the library where the output queue is located. This field can contain the special value *LIBL. If the output queue is *WRKSTN or *DEV, this field is blank.

Output queue library name. The name of the library the output queue is in.

Owner. This field indicates who is to own objects created by this user. The field contains one of the following values:

*USRPRF	The user owns any objects the user creates. If the user does not have a group profile, the field contains this value.

*GRPPRF The user's group profile owns any objects the user creates.

Password change date. The date the user's password was last changed, in *DTS (date-time stamp) format.

Password expiration interval. The number of days (from 1 through 366) the user's password can remain active before it must be changed. This field may contain one of the following special values:

0 The system value QPWDEXPITV is used to determine the user's password expiration interval.
-1 The user's password does not expire (*NOMAX).

Previous sign-on date and time. The date and time the user last signed on. The 13 characters are:

- CHAR(1): The century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
- CHAR(6): The date, in YYMMDD (year, month, day) format.
- CHAR(6): The time, in HHMMSS (hours, minutes, seconds) format.

Print device. The printer used to print for this user. This field can contain one of the following special values:

*WRKSTN The printer assigned to the user's work station is used.
*SYSVAL The default system printer specified in the system value QPRTDEV is used.

Reserved. An ignored field.

Set password to expire. Whether the user's password is set to expire, requiring the user to change the password when signing on. This field contains one of the following values:

Y The user's password is set to expire.
N The user's password is not set to expire.

Sign-on attempts not valid. The number of sign-on attempts that were not valid since the last successful sign-on.

Special authorities. The special authorities the user has. This field contains the following fields:

- CHAR(1): All object. Whether the user has all object special authority. If the user has the authority, this field is Y. If not, this field is N.
- CHAR(1): Security administrator. Whether the user has security administrator special authority. If the user has the authority, this field is Y. If not, this field is N.
- CHAR(1): Job control. Whether the user has job control special authority. If the user has the authority, this field is Y. If not, this field is N.
- CHAR(1): Spool control. Whether the user has spool control special authority. If the user has the authority, this field is Y. If not, this field is N.
- CHAR(1): Save system. Whether the user has save system special authority. If the user has the authority, this field is Y. If not, this field is N.
- CHAR(1): Service. Whether the user has service special authority. If the user has the authority, this field is Y. If not, this field is N.
- CHAR(9): Reserved.

Special environment. The special environment the user operates in after signing on. This field contains one of the following special values:

*SYSVAL The system value QSPCENV is used to determine the user's special environment.
*NONE The user operates in the OS/400 environment.
*S36 The user operates in the System/36 environment.

Status. The status of the user profile. This field contains one of the following values:

*ENABLED The user profile is enabled; therefore, the user is able to sign on.
*DISABLED The user profile is disabled; therefore, the user cannot sign on.

Storage used. This field contains the amount of auxiliary storage (in kilobytes) occupied by this user's owned objects.

Text description. The descriptive text for the user profile.

User class name. This field contains one of the following special values:

*SECOFR The user has a class of security officer.
*SECADM The user has a class of security administrator.
*PGMR The user has a class of programmer.
*SYSOPR The user has a class of system operator.
*USER The user has a class of end user.

User options. The options for users to customize their environment. This field contains the following fields:

- CHAR(1): Show keywords (*CLKWD). Whether the keywords are shown when a CL command is displayed. If the keywords are to be shown, this field is Y. If not, this field is N.
- CHAR(1): Show detailed information (*EXPERT). Whether more detailed information is shown when the user is defining or changing the system using edit or display object authority. This user option is independent of the ASTLVL parameter on the user profile and the ASTLVL parameter available on commands. If the details are to be shown, this field is Y. If not, this field is N.
- CHAR(1): Full screen help (*HLPFULL). Whether UIM online help is to be displayed on a full screen or in a window. If the full screen is to be shown, this field is Y. If not, this field is N.
- CHAR(1): Show status message (*STSMSG). Whether status messages sent to the user are shown. If the status messages are to be shown, this field is Y. If not, this field is N.
- CHAR(1): Do not show status message (*NOSTSMSG). Whether status messages sent to the user are not shown.
- CHAR(1): Roll key direction change (*ROLLKEY). Whether the opposite action from the system default

for roll keys is taken or not. If the opposite action is to be taken, this field is Y. If not, this field is N.
- CHAR(1): Printing complete message (*PRTMSG). Whether a message is sent to the user when a spooled file is printed or not. If a message is to sent to the user, this field is Y. If not, this field is N.
- CHAR(29): Reserved.

User profile name. The name of the user profile for which the information is returned.

Error Messages

CPF2203 E User profile &1 not correct.
CPF2225 E Not able to allocate internal system object.
CPF2299 E Not able to create internal security object.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.

Retrieve User Authority to Object (QSYRUSRA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Format name	Input	Char(8)
4	User name	Input	Char(10)
5	Qualified object name	Input	Char(20)
6	Object type	Input	Char(10)
7	Error code	I/O	Char(*)

The Retrieve User Authority to Object (QSYRUSRA) API returns a specific user's authority for an object to the call.

Authorities and Locks

The following authorities are required for the user calling this API, unless the user specified is *CURRENT, the caller owns the object, or the object is an authorization list:

- *OBJMGT authority to the object specified.
- *READ authority to the user profile specified (unless *PUBLIC is specified).

If previous programs in the program stack adopt their owner's authority, the adopted authority for the current program is the accumulated adopted authority from all other programs in the program stack that adopt authority.

When retrieving authority to logical files, no data authorities (read, add update, or delete) are returned. Logical files do not have data authorities associated with them; the data authorities are associated with the base physical file.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable used to return the user's authority to the object. This variable must be at least 8 bytes long.

Receiver variable length
INPUT; BINARY(4)
The length of the receiver variable. The variable must be at least 8 bytes long.

Format name
INPUT; CHAR(8)
The name of the format used to return the authority information. You can specify the following special value:

USRA0100 All authority information is returned. For a detailed description of this format, see "USRA0100 Format."

User name
INPUT; CHAR(10)
The name of the user whose object authority is returned. You can specify the following special values:

***CURRENT** The authority of the user currently running to the specified object is returned.
***PUBLIC** The public authority for the object is returned.

Qualified object name
INPUT; CHAR(20)
The name of the object whose authority is returned. The first 10 characters specify the object name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the object. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the object.

Object type
INPUT; CHAR(10)
The type of object for which authority information is returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Receiver Variable Description

The following tables describe the order and format of the data returned in the receiver variable. For detailed descriptions of the fields in the tables, see "Field Descriptions" on page 25-22.

USRA0100 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Object authority
18	12	CHAR(1)	Authorization list management
19	13	CHAR(1)	Object operational
20	14	CHAR(1)	Object management
21	15	CHAR(1)	Object existence
22	16	CHAR(1)	Data read
23	17	CHAR(1)	Data add
24	18	CHAR(1)	Data update
25	19	CHAR(1)	Data delete
26	1A	CHAR(10)	Authorization list
36	24	CHAR(2)	Authority source
38	26	CHAR(1)	Some adopted authority
39	27	CHAR(10)	Adopted object authority
49	31	CHAR(1)	Adopted authorization list management
50	32	CHAR(1)	Adopted object operational
51	33	CHAR(1)	Adopted object management
52	34	CHAR(1)	Adopted object existence
53	35	CHAR(1)	Adopted data read
54	36	CHAR(1)	Adopted data add
55	37	CHAR(1)	Adopted data update
56	38	CHAR(1)	Adopted data delete

Field Descriptions

Adopted authorization list management. Whether the user has adopted this authority to the object. If the user adopted the authority, this field is Y. If not, this field is N.

Adopted data add. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted data delete. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted data read. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted data update. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted object authority. The user's adopted authority to the object. This field is only valid if some of the user's authority is adopted. If the user does not adopt authority, this field and the other adopted authority fields will be blank. If the user adopts authority, this field contains one of the following values:

**ALL* The user adopted all object (operational, management, and existence) and data (read, add, update, and delete) authorities to the object.

**CHANGE* The user adopted object operational and all data authorities to the object.

**USE* The user adopted object operational and data read authorities to the object.

USER DEF The user adopted some combination of object and data authorities that do not relate to a special value. The individual authorities for the user should be checked to determine what authority the user has adopted to the object.

Adopted object existence. Whether the user adopted this authority to the object. If the user adopted the authority, this field is Y. If not, this field is N.

Adopted object management. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted object operational. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Authority source. Indicates where the authority that the user has to the object initially came from. The authority may be a combination of authority from this source plus adopted authority. This field contains one of the following special values:

UA The user has *ALLOBJ special authority.

UO The user is privately authorized to the object.

UL The user is privately authorized to the authorization list securing the object.

GA The user's group has *ALLOBJ special authority.

GO The user's group is privately authorized to the object.

GL The user's group is privately authorized to the authorization list securing the object.

PO The user accesses the object through the public authority.

PL The user accesses the object through the public authority on the authorization list securing the object.

AD All of the authority that the user has comes from adopted authority. This value is only returned if the user is *CURRENT.

Authorization list. The name of the authorization list securing the object. This field can contain one of the following special values:

**NONE* There is no authorization list securing the object.

**DAMAGED* The authorization list securing the object is damaged.

Authorization list management. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Bytes available. The number of bytes of data available to be returned to the user. If all data is returned, this is the same as the number of bytes returned. If the receiver variable was not big enough to contain all of the data, this is the number of bytes that can be returned.

Bytes returned. The number of bytes of data returned to the user. This is the lesser of the number of bytes available to be returned or the length of the receiver variable.

Data add. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data delete. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data read. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data update. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object authority. The special value indicating the user's authority to the object. This is the user's total authority to the object, including adopted authority (if the user is *CURRENT). This is one of the following values:

*ALL	The user has all object (operational, management, and existence) and data (read, add, update, and delete) authorities to the object.
*CHANGE	The user has object operational and all data authorities to the object.
*USE	The user has object operational and data read authorities to the object.
*EXCLUDE	The user has none of the object or data authorities to the object, or authorization list management authority.
USER DEF	The user has some combination of object and data authorities that do not relate to a special value. The individual authorities for the user should be checked to determine what authority the user has to the object.

Object existence. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object management. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object operational. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Some adopted authority. Whether some of the authority that the user has to the object comes from adopted authority. If some of the authority is adopted, this field is Y. If not, this field is N. This field can only contain Y if the user is *CURRENT.

Error Messages

CPF2203 E User profile &1 not correct.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF3C31 E Object type &1 is not valid.
CPF8122 E &8 damage on library &4.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9812 E File &1 in library &2 not found.
CPF9814 E Device &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.

Set Profile (QWTSETP) API

Parameters

Required Parameter:

1	Profile handle	Input	Char(12)

Optional Parameter:

2	Error code	I/O	Char(*)

The Set Profile (QWTSETP) API validates the profile handle, locks the user profile, and changes the job to run under the user and group profile represented by the profile handle. Once the change has been made, any open files and objects allocated by the original profile are accessible to the new profile.

No other attributes associated with the user or group profile are replaced. The qualified job name does not change to reflect the new user profile. However, any object created by the job while running under the new profile are owned by the new profile or its group profile.

If the profile handle is not valid, the QWTSETP API releases all profile handles previously created by the Get Profile Handle (QSYGETPH) API for the job, adds an exception to the job log, and enters a security violation in the QAUDJRN audit journal.

Required Parameter

Profile handle
INPUT; CHAR(12)
The profile handle returned by the QSYGETPH API for the user profile to switch the job to.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF22AD E Group profile for user not found.
CPF22E7 E Profile handle is not valid.
CPF2204 E User profile &1 not found.
CPF2213 E Not able to allocate user profile &1.
CPF2217 E Not authorized to user profile &1.
CPF3CF1 E Invalid error code structure specified.

Part 13. Spooled File APIs

Chapter 26. Spooled File APIs

Spooled file APIs obtain specific information about spooled files. For example, spooled file APIs can:

- Return a list of spooled files based on given selection criteria, such as a user or an output queue.
- Return information about the current status of an output queue.
- Provide functions to access a specific spooled file from which the API can return the attributes and data of a spooled file or create a duplicate of a specific spooled file.

Spooled file APIs are useful in writing applications to clean up, save, and restore spooled files.

The spooled file APIs are presented alphabetically. They include the following:

Close Spooled File (QSPCLOSP) closes a spooled file opened by the Open Spooled File (QSPOPNSP) API or created by the Create Spooled File (QSPCRTSP) API.

Create Spooled File (QSPCRTSP) creates a spooled file. The attributes for the spooled file are based on the values taken from the spooled file attributes parameter. When this spooled file is created, it does not contain any data.

Get Spooled File Data (QSPGETSP) gets data from an existing spooled file. The existing spooled file must have been opened by the Open Spooled File (QSPOPNSP) API. Data is retrieved from the existing spooled file by buffers (one or more) and stored in a user space. The data in the user space is used as source to the Put Spooled File Data (QSPPUTSP) API, which puts the data in the newly created spooled file.

List Spooled Files (QUSLSPL) generates a list of spooled files on the system into a user space. The list can include all spooled files or those of specific users, output queues, form types, or user-specified data values. The generated list replaces any existing lists in the user space.

Open Spooled File (QSPOPNSP) opens an existing spooled file. After the existing spooled file is opened, the Get Spooled File Data (QSPGETSP) API can then get the data and put it in the user space.

Put Spooled File Data (QSPPUTSP) puts data into a spooled file that was created using the Create Spooled File (QSPCRTSP) API. The data put in the spooled file is taken from a user space. The data in the user space can be created by either using the Get Spooled File Data (QSPGETSP) API or a user application.

Retrieve Job Queue Information (QSPRJOBQ) retrieves information associated with the specified job queue. Information returned includes the parameters used to create the queue, the current status of the queue, and the number of entries on the queue.

Retrieve Output Queue Information (QSPROUTQ) retrieves information associated with the specified output queue. Information returned includes the parameters used to create the queue, the current status of the queue, and the number of entries on the queue.

Retrieve Spooled File Attributes (QUSRSPLA) returns specific information about a spooled file into a receiver variable. The size of the receiver variable determines the amount of information returned. You can specify the spooled file for which information is returned either with the internal job and spooled file identifiers, or with a specific job name, spooled file name, and spooled file number.

QUSRTOOL and Spooled File APIs

A tool called save and restore spooled files (SAVRSTSPLF) is contained in library QUSRTOOL.

This tool uses most of the spooled file APIs discussed here. Additionally, some CL commands are used in the tool.

A description of the SAVRSTSPLF tool can be found in member TSRINFO, file QATTINFO, library QUSRTOOL. File QATTINFO also contains information on how to use QUSRTOOL.

Close Spooled File (QSPCLOSP) API

Parameters

Required Parameter Group:

1	Spooled file handle returned	Input	Binary(4)
2	Error code	I/O	Char(*)

The Close Spooled File (QSPCLOSP) API closes a spooled file opened by the Open Spooled File (QSPOPNSP) API or created by the Create Spooled File (QSPCRTSP) API. A handle, which identifies the spooled file, is returned by the QSPOPNSP and QSPCRTSP APIs. This handle is used as source to the QSPCLOSP API.

A **spooled file handle** is an internal number that identifies a particular spooled file. The system assigns the handle when an existing spooled file is opened using the QSPOPNSP API or when a new spooled file is created using the QSPCRTSP API. The handle is valid until the spooled file is closed or the job ends.

Required Parameter Group

Spooled file handle returned
INPUT; BINARY(4)
The handle returned by the QSPOPNSP API or QSPCRTSP API.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameters not valid
CPF33DF E Internal data area for opened spooled files destroyed.
CPF33D2 E Spooled file handle not valid.

Create Spooled File (QSPCRTSP) API

Parameters

Required Parameter Group:

1	Spooled file handle	Output	Binary(4)
2	Spooled file attributes	Input	Char(*)
3	Error code	I/O	Char(*)

The Create Spooled File (QSPCRTSP) API is used to create a new spooled file with attributes based on values taken from the spooled file attributes parameter. The field values of the spooled file attributes can be obtained using the Retrieve Spooled File Attributes (QUSRSPLA) API. The fields can also be defined by a user application.

Authorities and Locks

Special Authority
*SPLCTL. This authority is needed if you are creating a spooled file for another user.

Output Queue Authority
*USE

API QSPCRTSP Authority
PUBLIC(*EXCLUDE)

The API authority is set this way so that system administrators can control the use of this API. This API has security implications because you can create a spooled file from the data of another spooled file.

Required Parameter Group

Spooled file handle
OUTPUT; BINARY(4)
The handle to be used by subsequent put (QSPPUTSP API) and close (QSPCLOSP API) operations to identify the spooled file.

Spooled file attributes
INPUT; CHAR(*)
Attributes returned into the receiver variable by a prior Retrieve Spooled File Attributes (QUSRSPLA) API call with format SPLA0200 specified. These attributes are used to create the new spooled file. The current length of the attributes is 3301 bytes.

To see the SPLA0200 fields, go to "SPLA0200 Format Fields" on page 26-3.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Considerations Using the QSPCRTSP API

The following special considerations apply when using the QSPCRTSP API.

System Values: When using the QSPCRTSP API, all system values must be uppercase for the AS/400 system to recognize them.

New Spooled File Ownership: The spooled file that is created by the QSPCRTSP API is spooled under one of two jobs. The job is determined by the user name field. If the user name is the current user, it will be a part of the user's job and is owned by the user profile that the job was started with.

However, ownership of the new spooled file can be assigned to a different user by specifying a different user profile name in the user-name field. The current user must have *SPLCTL authority to assign the spooled file to another user. When this is done, the new spooled file will be owned by the user specified in the user-name field. The new spooled file is then part of a **special system job (QPRTJOB)** that is created for each user.

The user profile for the user name must already exist.

New Spooled File Placement: The new spooled file is placed on the output queue specified in the output queue name field. If the output queue is to correspond to the output queue of the user profile, the user application can use the command Retrieve User Profile (RTVUSRPRF) to retrieve the output queue name. The output queue name should be placed in the output queue fields before calling the QSPCRTSP API.

In both cases, the spooled file name will be the one contained in the spooled file attributes parameter. The spooled file number is the next sequential one available for the job that the spooled file becomes a part of.

In both cases (if you are in the System/36 environment), the system assigns a new System/36 spooled file identifier and a new internal spooled file identifier.

Many of the fields returned by the Retrieve Spooled File Attributes (QUSRSPLA) API are closely interrelated with other fields. Changing these field values can lead to unexpected results.

Error Messages: Error messages can be CPIxxxx or CPFxxxx. Error messages that start with CPI are informational and do not stop the job from running. These messages are stored in the job log.

Error messages that start with CPF cause the job to stop running. These messages are returned to the application program. The form in which they are returned is determined by the error code parameter.

Error checking on the fields includes:

Note: If a default is used, that value is obtained from the device file.

- Length of fields is equal to the length of format SPLA0200.
- Special values for fields

  When a value is specified that is not valid, message CPI33E2 is issued and the field default is used.
- Incorrect values for fields

  When the field specifies a system name and the name is not valid, message CPF33E2 is issued. In the other cases, message CPI33E2 is issued and the default for that field is taken.
- Incorrect combinations of fields

  Message CPF33E2 is issued when the value of one field must correspond to a specific value of another field.

SPLA0200 Format Fields

The following table presents, in the same order, all of the fields of format SPLA0200. It contains the following entries:

- Field

  This is the field of the format.
- Used

  This column indicates whether this field is used in describing a new spooled file.

  Y This field is used in describing a new spooled file. Users creating their own spooled file data streams must set the Y fields.

  N This field is not used in describing a new spooled file.
- Cross-dependency

  This column indicates whether this field is dependent on other fields.

  Y This field should be changed with extreme caution. This field is closely interrelated with other fields and could cause unpredictable results if not set correctly. The new value will be reflected in the newly created spooled file.

  N The field is easily changed, and this field is not related to or dependent on another field. The new value is reflected in the newly created spooled file.

  N/A This field is not used; therefore, there is no need to change it.
- Description

  This column indicates relationships between fields and contains a brief description of how to change the field.

Figure 26-1 (Page 1 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Bytes returned	Y	N	The total number of bytes contained in the field format. This must be the length of format SPLA0200.
Bytes available	N	N/A	
Format name	Y	N	The name of the format of the fields. The only accepted value is SPLA0200.
Internal job identifier	N	N/A	
Internal spooled file identifier	N	N/A	
Job name	N	N/A	
User name	Y	Y	The user name of the owner of the new spooled file being created. If the user name is the current user, the spooled file is created as part of the current job. If the user name is someone other than the current user, the spooled file is created as part of the special system job QPRTJOB for the user name specified. The current user must have *SPLCTL authority.
Job number	N	N/A	
Spooled file name	Y	N	The name to be assigned to the new spooled file.
Spooled file number	N	N/A	
Form type	Y	N	The type of form loaded in the printer to print the spooled file.
User-specified data	Y	N	The 10 characters of user-specified data that describe the file.
Status	N	N/A	

Figure 26-1 (Page 2 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
File available	Y	N	The time this spooled file becomes available to an output device for processing. The possible values are: ***IMMED** The spooled file is available as soon as it is opened. ***FILEEND** The spooled file is available as soon as it is closed. ***JOBEND** The spooled file is available when the job that created it has completed. When *JOBEND is specified and the spooled file is being created for another user, the value is changed to *FILEEND because the job it is assigned to is the special system job QPRTJOB. When a value is not one of the three listed, the default used is *FILEEND. In this case the default is not taken from the device file specified.
Hold file before written	Y	N	Whether the spooled file is to be held. The possible values are: ***YES** The file is held. ***NO** The file is not held.
Save file after written	Y	N	Whether the spooled file is saved after it is written. The possible values are: ***YES** Save the spooled file. ***NO** Do not save the spooled file.
Total pages	Y	N	The number of pages this printer file contains.
Page or record being written	N	N/A	
Starting page	Y	N	The page at which printing is to start for the spooled file. Special values include: **-1** Use the ending page value. **0** Printing starts on page 1.
Ending page	Y	N	The page at which printing is to end for the spooled file. 0 or 2147483647 indicates the last page. To print the entire spooled file, set this value to 0, 2147483647, or the last page number.
Last page printed	N	N/A	
Restart printing	Y	N	The number of the page where printing is restarted. The possible values are: **-1** *STRPAGE: The page specified in the starting page field. **-2,** *ENDPAGE: The last page. **restart page** The page number where printing is restarted. This number should be within the page range specified by the spooled file starting and ending page field.
Total copies	N	N	Total copies to be produced for this spooled file.
Copies left to produce	Y	N	The number of copies remaining to be produced on the printer. Valid values are 1 through 255.
Lines per inch	Y	Y	The number (in tenths) of lines per vertical inch defined in the printer file. The value 40 indicates 4 lines per inch. Valid lines per inch include 4, 6, 8, 9, and 12. Lines per inch of 3 and 7.5 are valid for double-byte character set devices.
Characters per inch	Y	Y	The number (in tenths) of characters per horizontal inch, defined in the printer file. The value 100 indicates 10 characters per inch. Valid characters per inch include 5, 10, 12, 15, and 16.7. Characters per inch of 13.3, 18, and 20 are valid for double-byte character set devices.
Output priority	Y	N	The priority of the spooled file. The priority ranges from 1 (highest) to 9 (lowest). The priority specified must be within the limits of the owner of the spooled file. The limit can be found by using the Retrieve User Profile (RTVUSRPRF) command, specifying the PTYLMT parameter.
Output queue name	Y	N	The name of the output queue where the new spooled file is placed. If the output queue specified in the user profile for the owner of the spooled file is to be used, the Retrieve User Profile (RTVUSRPRF) command can be used, specifying the OUTQ parameter. A blank output queue name defaults to the printer file.

Figure 26-1 (Page 3 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Output queue library name	Y	N	The name of the library that contains the output queue. The OUTQLIB parameter on the RTVUSRPRF can be used to retrieve the library name. A blank library name defaults to *LIBL.
Date file opened	N	N/A	
Time file opened	N	N/A	
Device file name	Y	Y	The name of the device file used to create the spooled file. The device file library name must also be specified.
Device file library name	Y	Y	The name of the library that contains the device file. The device file name must also be specified.
Program that opened file name	Y	Y	The name of the program that opened the spooled file. The library name of the program must also be specified.
Program library name that opened file	Y	Y	The name of the library that contains the program that opened the spooled file. The program name must also be specified.
Accounting code	Y	N	An identifier assigned by the system to record the resources used to write this spooled file. This can be retrieved for the user owning the spooled file by using the Retrieve User Profile (RTVUSRPRF) command, specifying the ACGCDE parameter.
Print text	Y	N	The text that is printed at the bottom of each page and on separator pages.
Record length	Y	Y	The length of spooled file records. If the field shows -1 (the special value *RCDFMT), this is an externally defined spooled file, and the length is included in the spooled file definition. The length includes the extra length of 1 for carriage control if it exists.
Maximum records	Y	N	The maximum number of records allowed in the spooled file when it is opened. Valid values are 1 through 500000. A special value of 0 indicates *NOMAX.
Device type	N	N	
Printer device type	Y	Y	The type of data stream used to represent the spooled file. The possible values are: ***AFPDS** Advanced Function Printing* data stream ***AFPDSLINE** AFPDS data mixed with 1403 line data ***IPDS** Intelligent printer data stream ***LINE** 1403 line data ***SCS** Systems Network Architecture (SNA) character stream ***USERACSII** ASCII data The data stream of the spooled file must correspond to the printer device type.
Document name	Y	Y	The name of the document that was the source of the spooled file. This field is used by the OfficeVision/400 program and must be blank for other spooled files. If the document name is specified, the folder name must also be specified.
Folder name	Y	Y	The name of the folder that contains the document name. This field is used by the OfficeVision/400 program and must be blank for other spooled files. If the folder name is specified, the document name must also be specified.
System/36 procedure name	Y	N	The name of the procedure running when the spooled file is created.
Print fidelity	Y	Y	The kind of error handling that is performed when printing spooled files with a printer device type of *AFPDS. The possible values are: ***CONTENT** The printing overrides errors in the data stream and continues printing with the printers best quality based on the content fidelity. ***ABSOLUTE** The spooled file is printed exactly as intended. Printing is stopped if an error is encountered in the data stream. This field should be set to *CONTENT for all non-AFPDS files.
Replace unprintable characters	Y	N	Whether characters that cannot be printed are to be replaced with another character. Valid values are Y (yes) or N (no).
Replacement character	Y	Y	The character that replaces any unprintable characters. This field has a value if the replace unprintable characters field specifies Y.

Figure 26-1 (Page 4 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Page length	Y	Y	The page length (in lines per page) used by the spooled file specified in the device file field. The valid range is row 1 through 255. The value should not exceed the actual length of the page.
Page width	Y	Y	The page width (in characters per printed line) used by the spooled file specified in the device file field. The valid range is column 1 through 378, although some printers have a page width less than 378. The value should not exceed the actual width of the page used.
Number of separators	Y	N	The number of file separator pages placed at the beginning of each copy. Valid values are 0 through 9, with 0 meaning no separator pages.
Overflow line number	Y	Y	The last line to be printed before the data being printed overflows to the next page. Valid values are 1 through 255. This value cannot be greater than the value specified in the page length field.
Double-byte character set (DBCS) data	Y	Y	Whether this spooled file contains double-byte character set (DBCS) data. The options are *YES or *NO. The other DBCS fields should also be set appropriately.
DBCS extension characters	Y	Y	Whether the system is to use the extension character processing function. The possible values are: ***YES** The system processes DBCS extension characters. ***NO** The system does not process DBCS extension characters; it prints extension characters as the undefined character. This field should be *NO when the double-byte character set (DBCS) data field is *NO.
DBCS shift-out shift-in (SO/SI) spacing	Y	Y	The presentation of shift-out and shift-in characters when printed. The possible values are: ***YES** Shift-out and shift-in characters occupy one space. ***NO** Shift-out and shift-in characters occupy no space. ***RIGHT** Shift-out characters occupy no space and shift-in characters occupy two spaces. This field should be *NO when the double-byte character set (DBCS) data field is *NO.
DBCS character rotation	Y	Y	Whether this printer file causes the DBCS characters to be rotated 90 degrees counterclockwise before printing. The possible values are *YES and *NO. This field should be *NO when the double-byte character set field is *NO.
DBCS characters per inch	Y	Y	The number of double-byte characters to be printed per inch. The possible values are: **-1** *CPI: One-half of the characters-per-inch value. **-2** *CONDENSED: 20 double-byte characters per 3 inches. **5** 5 characters per inch. **6** 6 characters per inch. **10** 10 characters per inch. This field should be -1 when the double-byte character set field is *NO.
Graphic character set	Y	Y	The set of graphic characters used when printing the spooled file. For *DEVD, the system gets the graphic character set from the printer device description. Valid values are 1 through 32767 and *DEVD. When a numeric value is specified, it is represented as a character string that is right-justified with leading zeros.
Code page	Y	Y	The mapping of graphic characters to code points for this printer. For *DEVD, the system gets the code page from the printer device description and this value is ignored. Valid values are 1 through 32767 and *DEVD. When a numeric value is specified, it is represented as a character string that is right justified with leading zeros.
Form definition name	Y	Y	The name of the form definition to use for printing the spooled file. The special value *DEVD is the default. See note 3 on page 26-18 for additional information about this field.
Form definition library name	Y	Y	The name of the library that contains the form definition. The special value *LIBL should be specified when the form definition name field is specified as *DEVD. See note 3 on page 26-18 for additional information about this field.

Figure 26-1 (Page 5 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Source drawer	Y	Y	The drawer to be used when the form feed field is set to *AUTOCUT. The possible values are: **1-255** The drawer number. **-1** *E1, the envelope drawer. **-2** *FORMDF, indicating that the file uses a user-specified form definition. This value is used only when the printer device type (DEVTYPE) field has been set to *LINE, *AFPDS, or *AFPDSLINE.
Printer font	Y	Y	The printer font that is used. The possible values are: ***DEVD** The font defined in the printer device description. ***CPI** A font that has the pitch specified by the characters-per-inch field. **identifier** The numeric font identifier that is used with the printer device file. The font specified should be valid for the printer.
System/36 spooled file identifier	N	N/A	
Page rotation	Y	N	The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. The possible values are: **-1** *AUTO: Computer output reduction is done automatically if the output is too large to fit on the form, regardless of the print quality. **-2** *DEVD: Page rotation is obtained from the printer device description. **-3** *COR: Output created for a form 13.2 inches wide by 11.0 inches long is adjusted to print on a form 11.0 inches wide by 8.5 inches long. **0** No rotation is done. Printing starts at the edge of the paper loaded into the printer first and is parallel to that edge. **90** Text is rotated 90 degrees clockwise from the 0-degree writing position. **180** Text is rotated 180 degrees clockwise from the 0-degree writing position. **270** Text is rotated 270 degrees clockwise from the 0-degree writing position.
Justification	Y	N	The percentage that the output is right-justified. The values are 100, 50, or 0.
Print on both sides (duplex)	Y	Y	How the information prints. The possible values are: ***FORMDF** A user-specified form definition determines how printing is done. This value is used only when the printer device type (on the printer file) is specified as *LINE, *AFPDS, or *AFPDSLINE. ***NO** The print appears on only one side of a piece of paper. ***YES** The print is on both sides of the page with the top of each page the same for both sides. ***TUMBLE** The print is on both sides of the page with the top of one printed page at the opposite end from the top of the other printed page.
Fold records	Y	N	Whether all positions in a record are displayed. The possible values are: ***YES** Records whose length exceeds the page width flow to the following line. ***NO** Records whose length exceeds the page width are truncated.
Control character	Y	Y	Whether the data to be printed used the American National Standard printer control character. The possible values are: ***NONE** No print control characters are passed in the data that is printed. ***FCFC** The first character of every record is an American National Standard printer control character. The record length field must include one position for forms control code. This value is not valid for externally described printer files.
Align forms	Y	N	Whether a forms alignment message is sent prior to printing the data. The options are *YES or *NO.
Print quality	Y	N	The print quality used when printing the data. The possible values are: ***STD** Standard ***DRAFT** Draft ***NLQ** Near-letter quality ***FASTDRAFT** Pages printed per minute are greater than if *DRAFT is specified

Figure 26-1 (Page 6 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Form feed	Y	N	The form feed attachment used by the printer device file. The possible values are: ***DEVD** The forms are fed into the printer as described in the device description. ***CONT** Continuous forms are used by the printer. The tractor-feed attachment must be put on the printer if this value is specified. ***CUT** Single-cut sheets are used by the printer. Each sheet is manually loaded. The forms alignment message is not sent for cut sheets. ***AUTOCUT** Single-cut sheets are semiautomatically fed into the printer. The sheet-feed attachment must be put on the printer if this value is specified. The forms alignment message is not sent for cut sheets.
Volumes (array)	N	N/A	Diskette information is not valid for these APIs.
File label identifier	N	N/A	Diskette information is not valid for these APIs.
Exchange type	N	N/A	Diskette information is not valid for these APIs.
Character code	N	N/A	Diskette information is not valid for these APIs.
Total records	Y	N	The number of data records. This field is only used when the device type (DEVTYPE) parameter of the printer file is *AFPDS, *LINE, or *AFPDSLINE.
Multiple up (pages per side)	Y	N	The number of logical pages that print on each side of each physical page when this spooled file prints. The possible values are 1, 2, and 4. See note 1 on page 26-18 for additional information about this field.
Front overlay name	Y	Y	The name of the **front overlay** (the material that prints on the front side of each page). The possible values are: ***NONE** The file does not use the front overlay. **front overlay name** The name of the front overlay. If an overlay name is specified, the front overlay library name must also be specified. See note 1 on page 26-18 for additional information about this field.
Front overlay library name	Y	Y	The name of the library containing the front overlay. The possible values are: ***CURLIB** The current library for the job locates the front overlay. ***LIBL** The library list locates the front overlay. **library name** This library is searched for the front overlay. See note 1 on page 26-18 for additional information about this field.
Front overlay offset down	Y	Y	The offset down from the point of origin where the overlay is printed. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 1 on page 26-18 for additional information about this field.
Front overlay offset across	Y	Y	The offset across from the point of origin where the overlay is printed. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 1 on page 26-18 for additional information about this field.
Back overlay name	Y	Y	The name of the **back overlay** (the material that prints on the back side of each page). The possible values are: ***FRONTOVL** The back overlay is the same as the front overlay. ***NONE** The file does not use a back overlay. **back overlay name** The name of the back overlay. If an overlay name is specified, the back overlay library name must also be specified. See note 1 on page 26-18 for additional information about this field.

Figure 26-1 (Page 7 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Back overlay library name	Y	Y	The name of the library containing the back overlay. The possible values are: ***CURLIB** The current library for the job locates the back overlay. ***LIBL** The library list locates the back overlay. **library name** This library is searched for the back overlay. See note 1 on page 26-18 for additional information about this field.
Back overlay offset down	Y	Y	The offset down from the point of origin where the overlay is printed. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 1 on page 26-18 for additional information about this field.
Back overlay offset across	Y	Y	The offset across from the point of origin where the overlay is printed. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 1 on page 26-18 for additional information about this field.
Unit of measure	Y	Y	The unit of measure to use for specifying distances. The unit of measure is used with certain parameters of the printer file. Page definitions and form definitions are examples of these parameters. The possible values are: ***CM** Centimeters ***INCH** Inches
Page definition name	Y	Y	The name of the page definition to use for the spooled file. This information is used only when the printer device type (DEVTYPE) field is *LINE or *AFPDSLINE. The special value of *NONE is used to mean no page definition. When a page definition name is given, the page definition library name must also be given.
Page definition library name	Y	Y	The name of the library in which the page definition resides. This information is necessary only when the printer device type (DEVTYPE) field is *LINE or *AFPDSLINE.
Line spacing	Y	Y	How a spooled file's line data records are spaced when printed. This information is necessary only when the printer device type (DEVTYPE) parameter on the printer file is *LINE or *AFPDSLINE. The possible values are: ***SINGLE** The printed output is single spaced. If the data contains control characters, they are ignored. ***DOUBLE** The printed output is double spaced. If the data contains control characters, they are ignored. ***TRIPLE** The printed output is triple spaced. If the data contains control characters, they are ignored. ***CTLCHAR** The control characters in the data determine the line spacing. This value should be used when the printer device type is not *LINE or *AFPDSLINE.
Point size	Y	Y	The point size in which this spooled file's characters should be printed. Valid values are 000.1 through 999.9. This field is ignored if the print font is specified as 0.0 (*DEVD).
Maximum spooled data record size	Y	Y	The length of the largest record in the spooled file. For DEVTYPE *LINE, *AFPDSLINE, and *AFPDS spooled files, this length is the length of the largest record in the spooled file. For all other printer device types (DEVTYPE), this length is the same as the spooled file buffer size.
Spooled file buffer size	Y	N	The length of the buffer being written. Sizes supported are 512 and 4079. A buffer size of 512 is recommended for spooled files that have a schedule field of *IMMED. A buffer size of 4079 is recommended for spooled files that have a schedule field of *FILEEND or *JOBEND. If the original spooled file was spooled with a buffer size of 4079, changing the buffer size to 512 causes a function check to occur.
Spooled file level	Y	N	The level of the spooled file in Version, Release, and Modification level format (VxRxMx).
Coded font array	Y	Y	The name of the fonts used when printing a spooled file with a printer device type (DEVTYPE) of *LINE or *AFPDSLINE. The name does not include the 2-character prefix of the coded font name (X0-XG). This array should be blank for all other printer device types.

Figure 26-1 (Page 8 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Channel mode	Y	Y	A variable indicating the channel value mode. They are: ***NORMAL** Use the normal channel values (1 equals skip to line 1, 2 through B equal space one line before printing, C equals skip to overflow line). **blank** The channel values specified in the channel value array are used.
Channel value array	Y	Y	The array contains 12 channels. Each channel is assigned a number, 1 through 12. Each channel (or number in the array) is assigned a value. That value is the line that the printer skips to before printing starts. For example, if channel 9 had a value of 9, the printer skips to line 9 before printing starts. When creating a file with data previously spooled and channel values specified, changing the channel values does not affect the newly created spooled file. The channel values used when printing are the ones specified when the data was originally spooled.
Graphics token	Y	Y	The printer type on which the graphics in the spooled file can be printed. The possible values are: **4214** Data stream contains graphics for a 4214 printer. **4234** Data stream contains graphics for a 4234 printer. **522X** Data stream contains graphics for a 522x printer. **IPDS** Data stream contains graphics for an IPDS printer.
Record format	Y	Y	The format of the records in the spooled file. The possible values are: ***FIXED** All records in the spooled file are the same size; that is, the original length specified in the print data for each record is the same for all records. ***VARIABLE** Records in the spooled file vary in size; that is, the original length specified in the print data for each record has at least one different value for records in the spooled file. For DEVTYPE *USERASCII, *IPDS, and *SCS without ASCII data files are all fixed format. *SCS files with ASCII data are variable length. For DEVTYPE *AFPDS, *AFPDSLINE, and *LINE, data files can be either fixed or variable.
Height of drawer 1	Y	Y	The height in inches of the paper in drawer 1. This field is set to packed 0 when the spooled file is being created as opposed to being copied. All other drawer fields should also be set to packed 0.
Width of drawer 1	Y	Y	The width in inches of the paper in drawer 1. This field is set to packed 0 when the spooled file is being created as opposed to being copied. All other drawer fields should also be set to packed 0.
Height of drawer 2	Y	Y	The height in inches of the paper in drawer 2. This field is set to packed 0 when the spooled file is being created as opposed to being copied. All other drawer fields should also be set to packed 0.
Width of drawer 2	Y	Y	The width in inches of the paper in drawer 2. This field is set to packed 0 when the spooled file is being created as opposed to being copied. All other drawer fields should also be set to packed 0.
Number of buffers	N	N/A	The number of buffers in the spooled file.
Maximum forms width	Y	N	The maximum width of printer file forms in character positions.
Alternate forms width	Y	Y	The width of the alternate forms in character positions. This field applies only to spooled files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields. The other alternate forms fields should also be set to 0.
Alternate forms length	Y	Y	The length of the alternate forms in lines. This field applies only to spooled files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields. The other alternate forms fields should also be set to 0.
Alternate lines per inch	Y	Y	The lines per inch for the alternate forms. This field applies only to spooled files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields. The other alternate forms fields should also be set to 0.
System/38 Text Utility flags	Y	N	The flag for advanced functions. This field is set to hexadecimal zeros when a spooled file is being created as opposed to copying an existing file.

Figure 26-1 (Page 9 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
File open	Y	N	Whether the spooled file is still open when the fields are retrieved. If fields retrieved from an open spooled file are used to create a new spooled file, the create operation fails. Changing this flag to N for a spooled file that is open could cause unpredictable results in a newly created spooled file.
Page count estimated	Y	Y	Whether the total pages given is an estimate. Valid values are Y (yes) or N (no).
File stopped printing on page boundary	Y	N	Whether the spooled file stops on a page boundary. Valid values are Y (yes) or N (no). If the spooled file has not been sent to a printer, set the field to N.
TRC for 1403	Y	Y	Whether the file contains 1403 line data with table-reference characters (TRC). This field is only used for device types (DEVTYPE) *LINE and *AFPDSLINE. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Define characters	Y	Y	Whether the spooled file defines or redefines unique print characters. For DEVTYPE(*SCS), a Load Alternate Characters command is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Characters per inch changes	Y	Y	Whether the spooled file changes the characters per inch (cpi) within the file. For DEVTYPE(*SCS), a Set Character Distance command is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Transparency	Y	Y	Whether the SCS Transparency command is used in the spooled file. It is set to N for all other device type files, such as IPDS. For DEVTYPE(*SCS), a TRANSPARENT SCS command is contained in the spooled file. Valid values are Y (yes) or N (no).
Double-wide character	Y	Y	Whether the spooled file prints everything twice as wide as normal. For DEVTYPE(*SCS), a Set Font Size command is contained in the data stream. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
DBCS character rotation command	Y	Y	Whether the double-byte character set characters are rotated 90 degrees counterclockwise before printing. • For DEVTYPE(*SCS), a Set Text Orientation command is contained in the spooled file. • For DEVTYPE(*AFPDS), a map coded font-1 (MCF-1) structured field that has the character rotation parameter set to hex 8700 is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Extended code page	Y	Y	Whether the extended code page changes within the file. • For DEVTYPE(*SCS), a Set CGCS through GCID is contained in the spooled file. • For DEVTYPE(*IPDS), a Load Font Equivalence command is inserted into the Load Font Equivalence table. • For DEVTYPE(*AFPDS), a map code font-2 (MCF-2) structured field is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
FFT emphasis	Y	Y	Whether the spooled file contains text that is to appear darker than the surrounding text. For DEVTYPE(*SCS), Begin Emphasis and End Emphasis commands are contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
3812 SCS	Y	Y	Whether the spooled file contains fonts supported only on the 3812 printer. For DEVTYPE(*SCS), a Set Coded Font Local command with a global ID of greater than 255 is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Set line density command	Y	Y	Whether the lines per inch (lpi) changes within the spooled file or is specified with the Set Line Density SCS command. For DEVTYPE(*SCS), a Set Line Density command is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.

Figure 26-1 (Page 10 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Graphics error actions	Y	Y	Whether the file contains graphic error action commands. For SCS files, the file contains one or more Set Graphic Error Action commands. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
5219 commands	Y	Y	Whether the file contains commands that are not valid on a 5219 printer. The data stream contains one or more of the following SCS commands, which either are not valid on a 5219 or contain parameters that are not valid. • An incorrect Set Character Set Global ID command is found by validity checking the code page with what is valid for the 5219 printer. • A Begin and End Emphasis SCS commands. • A Set Presentation Page Size command contains a page size value that is not supported. • A Page Presentation Media command contains a drawer value other than drawer 1 or drawer 2. • A Set Single Line Distance command contains a distance value that is not supported. • A Set Single Line Spacing command contains a distance value that is not supported. • A Justify Text Field Format command contains a value other than 0, 50, or 100 percent. • A Set Justify Mode Format command contains a value other than 0, 50, or 100 percent. • A Set Presentation Color command. • A Begin Overstrike command that contains an optional CHRID value other than 0. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
3812 SCS commands	Y	Y	Whether the file contains commands that are not valid on the 3812 printer. • An incorrect Set Character Set Global ID command is found by validity checking the code page with what is valid for the 3812 printer. • An incorrect Set Character Set Local ID command is found by validity checking the code page with what is valid for the 3812 printer. • A Set Presentation Color command Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Field outlining	Y	Y	Whether the spooled file outlines fields of data with boxes. If it does, the file must be printed on a printer that supports field outlining. For DEVTYPE(*SCS), a Define Grid Line command is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.

Figure 26-1 (Page 11 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Final form text	Y	Y	Whether this spooled file contains various functions that are supported on letter-quality printers. • For DEVTYPE(*SCS), a Document Content Architecture (DCA) command is contained in the spooled file. DCA commands include: – Required New Line – Required Form Feed – Indent Tab – Set Presentation Page Size – Set Horizontal Margins – Set Vertical Margins – Release Left Margin – Set Line Spacing – Set Single Line Distance – Justify Text Field Format – Set Justify Mode – Set Horizontal Tab Stops – Set Indent Level – Set Exception Action – Set Presentation Color – Set Spacing Variable – Begin and End Overstrike – Begin and End Underscore – Bell – Switch – Repeat – Tab – Backspace – Unit Backspace – Substitute – Word Underscore Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Bar code	Y	Y	Whether the spooled file contains bar codes created using the BARCODE keyword in the data description specifications (DDS). For DEVTYPE(*IPDS and *AFPDS), a series of bar code commands is contained in the data stream. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Color	Y	Y	Whether an IPDS Write Text command or AFPDS presentation text data (PTX) structured field containing a Set Text Color control is contained in the spooled file. For DEVTYPE(*IPDS and *AFPDS), a Set Text Color text control is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Drawer change	Y	Y	Whether the printer drawer is changed within the spooled file. • For DEVTYPE(*SCS), a Page Presentation Media command is contained in the spooled file. • For DEVTYPE(*IPDS), an Execute Order Home State-Select Input Media Source command is in the spooled file. • For DEVTYPE(*AFPDS), a media map and an invoke media map are in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Character ID	Y	Y	Whether the character ID can change within the file. • For DEVTYPE(*SCS), a Set CGCS through Local ID command is contained in the spooled file. • For DEVTYPE(*IPDS), a Load Font Equivalence command is inserted into the Load Font Equivalence table. • For DEVTYPE(*AFPDS), a map code font-2 (MCF-2) structured field is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.

Figure 26-1 (Page 12 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Lines per inch changes	Y	Y	Whether the lines per inch (lpi) changes within the spooled file. • For DEVTYPE(*SCS), a Set Single Line Density command is contained in the spooled file. • For DEVTYPE(*IPDS), a Load Page Descriptor command is contained in the spooled file, which specifies the line-per-inch value • For DEVTYPE(*AFPDS), a presentation text description (PTD) structured field that specifies the baseline increment is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Font	Y	Y	Whether the spooled file uses multiple fonts. • For DEVTYPE(*SCS), a Set Font Global command is in the spooled file. • For DEVTYPE(*IPDS), a Load Font Equivalence command is contained in the Load Font Equivalence table. • For DEVTYPE(*AFPDS), a map coded font-1 (MCF-1) or map coded font-2 (MCF-2) structured field is in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Highlight	Y	Y	Whether the spooled file contains text that is to appear darker than the surrounding text. • For DEVTYPE(*SCS), data is overprinted to get the bold effect. • For DEVTYPE(*IPDS), a Load Font Equivalence command is contained in the Load Font Equivalence table, which has the bold field set on. • For DEVTYPE(*AFPDS), a map coded font-2 (MCF-2) structured field that has the font weight class parameter set to hex 07 is in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Page rotate	Y	Y	Whether the spooled file changes the page rotation to be used within the file. • For DEVTYPE(*SCS), a Set Text Orientation command is contained in the spooled file. • For DEVTYPE(*IPDS), a Load Page Descriptor command is contained in the spooled file. See the *Intelligent Printer Data Stream Reference* for detailed information about rotating text in a spooled file. • For DEVTYPE(*AFPDS), a presentation text descriptor (PTD) structured field that specifies the page rotation is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Subscript	Y	Y	Whether the spooled file contains subscript. • For DEVTYPE(*SCS), a Subscript command is contained in the spooled file. • For DEVTYPE(*IPDS and *AFPDS), a Temporary Baseline Move text control is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Superscript	Y	Y	Whether the spooled file contains superscript. • For DEVTYPE(*SCS), a Superscript command is contained in the spooled file. • For DEVTYPE(*IPDS and *AFPDS), a Temporary Baseline Move text control is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
DDS	Y	Y	Whether the file was constructed with DDS. Valid values are Y (yes) or N (no). This field should be set to N when the file contains data that has not been spooled before.
Final form feed	Y	Y	Whether the Final Form Feed command is in the printer file. Valid values are Y (yes) or N (no).
SCS data	Y	Y	Whether the spooled file is created with SCS already in the input data. Valid values are Y (yes) or N (no).
User-generated data stream	Y	Y	Whether the data stream has been validated by a system program on the AS/400 system when the data was spooled, for example, the OfficeVision/400 program. Valid values are Y (yes) or N (no). This field should be set to Y (data stream not validated) when opening a spooled file that contains data that has not been spooled, for example, OfficeVision/400 spooled files.

Figure 26-1 (Page 13 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Graphics	Y	Y	Whether the spooled file contains graphics commands in its data stream. Valid values are Y (yes) or N (no).
Unrecognizable data	Y	Y	Whether the file contains SCS commands that are not valid because of one of the following: • Command is not recognized • Command data is not valid • Command is recognized, but not supported Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
ASCII transparency	Y	Y	For SCS files, whether ASCII commands are embedded in the ASCII transparency command. The ASCII transparency command is command_ID/command_length/ASCII_data. The command ID is a 1-byte field with the value hex 03. The command length is a 1-byte field that contains the length of the command length field and the ASCII data field. Valid values are Y (yes) or N (no). This field should be set to N when creating a spooled file as opposed to copying an existing spooled file.
IPDS transparent data	Y	N	Whether the file contains data from System/36 Programming Request for Price Quotations (PRPQs). Valid values are Y (yes) or N (no). This field should be set to N when creating a spooled file as opposed to copying an existing spooled file.
OfficeVision/400	Y	N	Whether the spooled file is generated by the OfficeVision/400 licensed program. Valid values are Y (yes) or N (no). This field should be set to N when creating a spooled file as opposed to copying an existing spooled file.
Lines-per-inch (lpi) value not supported	Y	N	This field applies only to OfficeVision/400 files. Valid values are Y (yes) or N (no). It will be N for all other spooled files. The lines-per-inch (lpi) value is designed to support 1440 lines per inch. To be able to visually read any data, you would have to select a value that creates a fraction that represents the number of lines printed per inch. For example, if the lpi value were 100, you would get 14.4 lines per inch (1440 divided by 100).
CPA3353 message	Y	N	Whether message CPA3353 is issued when this spooled file is printed. Valid values are Y (yes) or N (no).
Set exception	Y	Y	Whether the spooled file has the SCS Set Exception Action command in the data stream. Valid values are Y (yes) or N (no).
Carriage control character	Y	Y	Whether the data contains carriage control characters. This field is valid only when the printer device type (DEVTYPE) field is *LINE or *AFPDSLINE. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Page position	Y	Y	Whether page positioning errors are reported. This field is valid only when the printer device type (DEVTYPE) field is *LINE, *AFPDSLINE, or *AFPDS. Valid values are Y (yes) or N (no). This field is set to N for all other printer device types.
Character not valid	Y	Y	Whether incorrect character errors are reported. This field is valid only when the printer device type (DEVTYPE) field is *LINE, *AFPDSLINE, or *AFPDS. Valid values are Y (yes) or N (no). This field is set to N for all other printer device types.
Lengths present	Y	Y	Whether the 8-byte length information is present in the print text data buffers for format SPLF0200. This field is valid when the printer device type (DEVTYPE) field is *AFPDS, *AFPDSLINE, or *LINE. Valid values are Y (yes) or N (no). It must be set to Y for *AFPDSLINE and *LINE printer device types. *AFPDS data may or may not have the length information. This field should be set to N for all other printer device types.
5A present	Y	Y	Whether the hex 5A constant is present in all AFPDS data for the file (in format SPLF0200). This field is valid when the printer device type (DEVTYPE) field is *AFPDS. Valid values are Y (yes) or N (no). This field should be set to N for all other printer device types.

Figure 26-1 (Page 14 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Number of font array entries	Y	Y	The number of font equivalence array entries used. The valid values for the field are 0 through 48, where 0 indicates that there are no entries in the array. The field is valid only for spooled files that have the printer device type field equal to *IPDS. It should be set to zero for all other printer device types.
Number of resource library entries	N	N	The number of entries in the resource library array used. The valid values for the field are 0 through 43, where 0 indicates that there are no entries in the array.
Font equivalence array	Y	Y	The data portion of the Load Font Equivalence (LFE) command for intelligent printer data streams (IPDS). The current maximum is 48. However, for future expansion, the array returns 72 sixteen-character font equivalences. To support that further expansion, the 1153rd character of the variable is set to 1 when more than 72 entries are available. The format of this entry is described in the *Intelligent Printer Data Stream Reference*.
Resource library array	N	N	The library names in the library list to use when the spooled file is printed. The current maximum is 43. However, for future expansion, the resource library array returns 63. The additional space is provided for further expansion. To support that further expansion, the 631st character of the variable is set to 1 when more than 63 entries are available.
AS/400-created AFPDS	Y	Y	Whether the spooled file contains AFPDS data created on the AS/400 system. Valid values are Y (yes) or N (no).
Front margin offset down	Y	Y	For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) field), how far down from the top of the page printing starts. The possible values are: **-2** *DEVD. For printers configured AFP(*YES), no print border is used for front margin offsets across and down. Otherwise, the offset values are 0. **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 2 on page 26-18 for additional information about this field.
Front margin offset across	Y	Y	For the front side of a piece of paper it, specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 2 on page 26-18 for additional information about this field.
Back margin offset down	Y	Y	For the back side of a piece of paper, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are: **-1** The back margin offsets are the same as the front margin offsets. **-2** *DEVD. For printers configured AFP(*YES), no print border is used for back margin offsets across and down. Otherwise, the offset values are 0. **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 2 on page 26-18 for additional information about this field.
Back margin offset across	Y	Y	For the back side of a piece of paper, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far from the left side of the page printing starts. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 2 on page 26-18 for additional information about this field.
Length of page	Y	Y	The length of a page in rows and columns (*ROWCOL) or in inches and centimeters (*UOM). If this field is specified and the measurement method used is *ROWCOL, this value should be the same as that specified on the page length field found in this format. If this field is specified and the measurement method is *UOM, the actual distance specified must be equivalent to that specified on the page length field found in this format. If the measurement method is *ROWCOL, the page length value must be from 1 through 255. If the measurement method is *UOM, the page length value must be between 0 and 57.79.

Figure 26-1 (Page 15 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Width of page	Y	Y	The width of a page in rows and columns (*ROWCOL) or in inches and centimeters (*UOM). The width of a page is determined by the value specified on the PAGESIZE parameter of the printer file. If this field is specified and the measurement method used is *ROWCOL, this value should be the same as that specified on the page width field found in this format. If this field is specified and the measurement method is *UOM, this value must be equivalent to that specified on the page width field found in this format. If the measurement method is *ROWCOL, the page width value must be from 1 through 378. If the measurement method is *UOM, the page width value must be between 0 and 57.79.
Measurement method	Y	Y	The measurement method used for the length of page and width of page fields. The possible values are: ***ROWCOL** Uses rows and columns as the units of measure. ***UOM** Uses the value specified on the unit of measurement parameter (UOM). UOM is either inches (*INCH) or centimeters (*CM).
Advanced Function Printing (AFP*) resource	Y	Y	Whether this spooled file refers to AFP resources (using DDS) external to this spooled file, for example, page segments or overlays. The possible values are: **Y** The spooled file refers to AFP resources external to the spooled file. **N** The spooled file does not refer to AFP resources external to the spooled file.
Character set name	Y	Y	The name of the font character set object used to print this file. The possible values are: ***FONT** The information specified on the font parameter is used instead of the character set and code page. **character set name** The name of the font character set object to use. See note 4 on page 26-18 for additional information about this field.
Character set library name	Y	Y	The name of the library containing the font character set object. The possible values are: ***LIBL** The library list is used to locate the font character set object. **library name** This library is searched for the font character set object. See note 4 on page 26-18 for additional information about this field.
Code page name	Y	Y	The name of the code page used to print this spooled file. See note 4 on page 26-18 for additional information about this field.
Code page library name	Y	Y	The name of the library containing the code page used to print this spooled file. The possible values are: ***LIBL** The library list is used to locate the code page. **library name** This library is searched for the code page. See note 4 on page 26-18 for additional information about this field.
Coded font name	Y	Y	The name of the coded font used to print this spooled file. A **coded font** is an AFP resource composed of a font character set and a code page. The possible values are: ***FNTCHRSET** The values used are the values specified on the character set and code page fields. **coded font name** The name of the coded font used to print this spooled file. See note 4 on page 26-18 for additional information about this field.
Coded font library name	Y	Y	The name of the library containing the coded font used to print this spooled file. The possible values are: ***LIBL** The library list is used to locate the coded font. **library name** This library is searched for the coded font. See note 4 on page 26-18 for additional information about this field.
DBCS-coded font name	Y	Y	Specifies the name of the DBCS-coded font used to print DBCS data on printers configured as AFP(*YES). The possible values are: ***SYSVAL** The DBCS-coded font specified in the system value is used. **DBCS-coded font name** The name of the DBCS-coded font being used. See note 5 on page 26-18 for additional information about this field.

Figure 26-1 (Page 16 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
DBCS-coded font library	Y	Y	The name of the library containing the DBCS coded font used to print this spooled file. The possible values are: ***CURLIB** The current library is searched for the DBCS-coded font. ***LIBL** The library list is used to locate the DBCS-coded font. **Library name** This library is searched for the DBCS-coded font. See note 5 on page 26-18 for additional information about this field.
User-defined file	N	N	Whether the spooled file was created by using an API. The possible values are: ***YES** The spooled file was created using the QSPCRTSP API. ***NO** The spooled file was created by normal system functions.

Notes:

1 This field should only be specified for spooled files with device types of *SCS, *IPDS, or *AFPDS if the spooled file was created on an AS/400 system. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

2 This field should only be specified for spooled files with device types of *LINE, *AFPDSLINE, or *AFPDS if the spooled file was created on an AS/400 system. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

3 This field should only be specified for spooled files with device types of *LINE, *AFPDSLINE, or *AFPDS if the spooled file was not created on an AS/400 system. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

4 This field should only be specified for spooled files that contain *AFPDS created on the AS/400 system. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

5 This field should only be specified on a DBCS system, and only for spooled files that will be created with a device type of *SCS or that will contain AS/400-created *AFPDS. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 is not valid.
CPF33DD E Maximum number of open spooled files exceeded for this job.
CPF33DE E Size of internal data area for opened spooled file exceeds maximum.
CPF33DF E Internal data area for opened spooled files destroyed.
CPF33E0 E Incomplete set of attributes provided.
CPF33E1 E Attributes are for an opened spooled file.
CPF33E2 E Value &1 for spooled file attribute at offset &2 not valid.
CPF34B1 E Spooled file print data format not supported.
CPF34B2 E Spooled file input record length is not valid.
CPF9838 E User profile storage limit exceeded.
CPF9845 E Error occurred while opening spooled file &1.
CPI33E2 I Value &1 for spooled file attribute at offset &2 not valid.
CPI8098 I Error on distribution of spooled file to user &3 and &4.

Get Spooled File Data (QSPGETSP) API

Parameters

Required Parameter Group:

1	Spooled file handle	Input	Binary(4)
2	User space name and library	Input	Char(20)
3	Format name	Input	Char(8)
4	Ordinal number of buffer to be read	Input	Binary(4)
5	End of open spooled file	Input	Char(10)
6	Error code	I/O	Char(*)

The Get Spooled File Data (QSPGETSP) API gets data from an existing spooled file. The existing spooled file must have been previously opened by the Open Spooled File (QSPOPNSP) API. Data is retrieved from the existing spooled file by buffers (one or more) and stored in a user space. The data in the user space is used as source to the Put Spooled File Data (QSPPUTSP) API. The number of buffers returned in the user space is no greater than the value specified on the number of buffers to get parameter on the Open Spooled File (QSPOPNSP) API. The user space is automatically extended, if necessary, to allow all buffers requested to be stored in the user space.

User spaces can be created using the Create User Space (QUSCRTUS) API. The time it takes to create a user space is decreased if you specify to initialize the space to hexadecimal zeros.

Note: The size of the buffer (currently 512 or 4079 bytes) is determined by the buffer size of the existing spooled file being worked with.

Authorities and Locks

User Space Authority	*CHANGE
Library Authority	*USE
Output Queue Authority	*USE
User Space Lock	*EXCLRD

Required Parameter Group

Spooled file handle
INPUT; BINARY(4)
The handle returned by the Open Spooled File (QSPOPNSP) API.

User space name and library
INPUT; CHAR(20)
The name of the user space that is to receive the buffer of spooled information. The first 10 characters contain the user space name and the second 10 characters contain the name of the library in which the user space is located. The special values allowed for the library name are *LIBL and *CURLIB. Both user space name and library name are left-justified. If no library is specified as the current library of the job, QGPL is used.

Format name
INPUT; CHAR(8)
The format and the content of the information retrieved from each buffer. The possible values are:

SPFR0100 Information about the print data stored in the buffer.
SPFR0200 Information about the print data stored in the buffer and the print data.

Ordinal number of buffer to be read
INPUT; BINARY(4)
The buffer of the spooled file that is read first (or next).

Any number greater than zero is valid. When a specific buffer is requested, only that buffer is returned. The following special value is supported for this parameter:

-1 Reads the next buffer in the spooled file.

To read sequentially, starting at a specific buffer, the first API call should specify the specific buffer, with subsequent calls of the API specifying the special value -1 (read next). The first API call returns only the specific buffer requested. The subsequent read operations, with -1 specified for the next buffer, return the number of buffers specified on the open operation. When reading an entire spooled file, the special value -1 should be used.

Performance degradation could happen if a specific buffer is always used.

End of open spooled file
INPUT; CHAR(10)
How to handle the situation where the spooled file has not been closed (by this job or another job) and the requested data has not yet been written by the application program. The values supported for this parameter are:

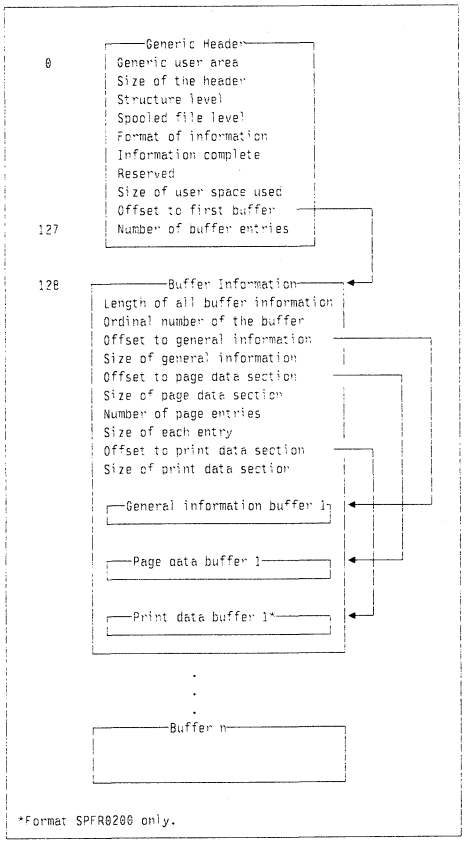
***WAIT** The API waits until the requested data is available or until the spooled file is closed. If the spooled file is closed without the requested data becoming available, the action defined by *ERROR is done.
***ERROR** The API returns either an error code or an error message based on what was specified for the error code parameter.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the User Space

The organization of the user space is dependent on the format (SPFR0100 or SPFR0200) used. If SPFR0100 is used, no print data is returned.

The following diagram shows the general structure of the user space. Offset values are calculated from the beginning of the user space.



Generic Header Section

The following table shows the generic header information returned for the SPFR0100 and SPFR0200 formats.

For more details about the fields, see "Field Descriptions" on page 26-20.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(64)	Generic user area
64	40	BINARY(4)	Size of header
68	44	CHAR(4)	Structure level
72	48	CHAR(6)	Spooled file level
78	4E	CHAR(8)	Format of the information returned
86	56	CHAR(1)	Information complete indicator
87	57	CHAR(1)	Reserved
88	58	BINARY(4)	Size of user space used
92	5C	BINARY(4)	Offset to first buffer
96	60	BINARY(4)	Number of buffers requested
100	64	BINARY(4)	Number of buffers returned
104	68	CHAR(24)	Reserved

Field Descriptions

Format of the information returned. The SPFR0100 or SPFR0200 format of the data returned.

Generic user area. Provided for the user's application program.

Information complete indicator. Used to indicate if all information requested has been supplied.

I Incomplete information. An interruption causes the user space to contain incomplete information about a buffer or buffers.

P Partial and accurate information. Partial information is returned when the maximum user space was used and not all of the buffers requested were read.

C Complete and accurate information. All the buffers requested are read and returned. Complete also applies to the case where the end of file was reached on a closed spooled file, but the number of buffers requested was not reached.

Number of buffers requested. The number of buffer entries requested for each read operation.

This is the number specified on the open operation. However, if a specific buffer is requested by the read operation, this value is only one buffer.

Number of buffers returned. The number of buffer entries returned.

This number may be equal to or less than the number of buffers requested. The number may be less because the end of a closed spooled file is reached or the maximum user space has been used before the requested buffer count has been processed.

Offset to first buffer. Locates the data for the first buffer returned. The offset value is from the beginning of the user space.

Reserved. Used to align a 4-byte boundary.

Size of header. The size of the header section of format SPFR0100 or SPFR0200 in bytes.

Size of user space used. The number of bytes used (returned) in the user space. This includes the user area, header, general information section, page section, and print data section.

Spooled file level. The level of the spooled file in Version, Release, Modification level format.

Structure level. The level of the structure of the user space. This should always be 0200.

Buffer Information Section

The following buffer information is returned by the QSPGETSP API. For more details about the fields in the following table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Length of all buffer information
4	4	BINARY(4)	Ordinal number of the buffer
8	8	BINARY(4)	Offset to general information section
12	C	BINARY(4)	Size of general information section
16	10	BINARY(4)	Offset to page data section
20	14	BINARY(4)	Size of page data section
24	18	BINARY(4)	Number of page entries
28	1C	BINARY(4)	Size of page entry
32	20	BINARY(4)	Offset to print data section
36	24	BINARY(4)	Size of print data section

Field Descriptions

Length of all buffer information. The size of all information pertaining to this buffer. This includes the general information, page data, and print data.

Number of page entries. The number of page entries in the page data section.

Offset to general information section. The location of the general information section. The offset value is from the beginning of the user space.

Offset to page data section. The location of the page data information section. The offset value is from the beginning of the user space.

Offset to print data section. The location of the print data section. The offset value is from the beginning of the user space.

Ordinal number of the buffer. The ordinal number of the buffer returned.

Size of general information section. The size of the general information section in bytes.

Size of page data section. The size of the page data section in bytes.

Size of page entry. The size of each page entry.

Size of print data section. The size of the print data section in bytes. This is zero (0) for format SPFR0100 because this format does not return this information. For data not previously spooled (creating as opposed to copying) the print data buffer section cannot exceed the size of the spooled file buffer minus the sum of the sizes of the following:

- Constant of 24
- Size of page data section

General Information Section

The general information section of formats SPFR0100 and SPFR0200 has the following structure.

For more details about the fields in the following table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Nonblank lines in buffer
4	4	BINARY(4)	Nonblank lines in first page
8	8	BINARY(4)	Buffer number of error information
12	C	BINARY(4)	Offset to error recovery information
16	10	BINARY(4)	Size of print data
20	14	CHAR(10)	State
30	1E	CHAR(1)	Last page continues
31	1F	CHAR(1)	Advanced print function file
32	20	CHAR(1)	LAC command array in buffer
33	21	CHAR(1)	Any buffer had LAC
34	22	CHAR(1)	Error recovery information contains LAC
35	23	CHAR(1)	Error recovery information
36	24	CHAR(1)	Zero pages
37	25	CHAR(1)	Load font
38	26	CHAR(1)	IPDS data
39	27	CHAR(5)	Reserved

Field Descriptions

Format SPFR0200 is the format required by the Put Spooled File Data (QSPPUTSP) API. When a spooled file is being copied, the information is retrieved with the QSPGETSP API. When the spooled file is created by a user application, all information in the format must be accurately provided.

Advanced print function file. Whether this spooled file was created by the Advanced Print Function Utility/400. The value is N if the device type is *SCS.

Any buffer had LAC. Whether any buffer had an SCS Load Alternate Characters (LAC) command since last error recovery information was saved. Specify Y for yes and N for no. This should be set to N for all spooled files with a DEVTYPE other than *SCS.

Buffer number of error information. Whether any buffer had an SCS LAC command since the last error recovery information was saved. Specify Y for yes and N for no. This should be set to N for all spooled files with DEVTYPE other than SCS.

Error recovery information. Whether there is error recovery information in this buffer for SCS commands. This error recovery information allows a file to be repositioned and print only some pages of a file, making sure the characters print using the proper code point. Information on the proper code points is stored in LAC commands in the error recovery information. This information is stored for each page that contains SCS commands that can cause characters to print differently based on the code point specified. Specify Y for yes and N for no. This should be set to N for all spooled files with a device type other than SCS.

Error recovery information contains LAC. Error recovery information for SCS commands at the end of this page includes the LAC command found later in the buffer. Specify Y for yes and N for no. This should be set to N for all spooled files with a DEVTYPE other than SCS.

IPDS data. Whether the buffer contains IPDS data. Specify Y for yes and N for no. Buffers of IPDS data may exist in SCS spooled files.

LAC command array in buffer. An LAC command exists in this buffer. Specify Y for yes and N for no. This should be set to N for all spooled files with a DEVTYPE other than SCS.

Last page continues. The last page in this buffer continues in the next buffer when the value is Y.

Load font. Indicates whether a new load font equivalence entry was added to the table for this buffer. Specify Y for yes and N for no. This should be set to N for all spooled files with a device type other than IPDS.

Nonblank lines in buffer. The number of nonblank lines in the buffer.

Nonblank lines in first page. The number of nonblank lines in the first page ending in the buffer.

Offset to error recovery information. Offset to the error recovery information in this buffer. This field is 0 for all spooled files with a DEVTYPE other than SCS.

Reserved. Reserved for byte alignment.

Size of print data. The number of bytes of print data.

State. The state the buffer ends in if the data is IPDS.

**HOME*	Home state
**PAGE*	Page state
**GRAPHICS*	Graphics
**PAGETRANS*	Page transparency
**HOMETRANS*	Home transparency
blank	All non-IPDS files

Zero pages. Whether there are zero (0) pages in the spooled file. Specify Y for yes and N for no.

Page Data Section

Each entry of the page data section of format SPFR0100 and SPFR0200 has the following structure. For more details about the fields in the following table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Text data start
4	4	BINARY(4)	Any data start
8	8	BINARY(4)	Page offset

Field Descriptions

Any data start. The number of the first line where user data can start on this page. This count includes all data to be printed.

Page offset. The location of the start of this page. The offset value is from the beginning of the print data.

Text data start. Number of the first line where user data can start on this page. This count includes text only.

Print Data Section

The print data section of format SPFR0200 has the following structure. For more details about the fields in the following tables, see "Field Descriptions" on page 26-23.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(*)	Print data

The format of the print text data is dependent on the printer device type. For printer device types of *AFPDS, *AFPDSLINE, and *LINE, the data for a given format may span buffers. The first buffer returned will have one of the following formats:

- Printer device type of *LINE with the control character field set to *NONE and the TRC field set to N.

Original length	Trimmed length	Line number	DBCS flag	RSRVD	Line data

- Printer device type of *LINE with the control character field set to *FCFC and the TRC field set to N.

Original length	Trimmed length	Line number	DBCS flag	RSRVD	CC	Line data

- Printer device type of *LINE with the control character field set to *NONE and the TRC field set to Y.

Original length	Trimmed length	Line number	DBCS flag	RSRVD	TRC	Line data

- Printer device type of *LINE with the control character field set to *FCFC and the TRC field set to Y.

Original length	Trimmed length	Line number	DBCS flag	RSRVD	CC	TRC	Line data

- Printer device type of *AFPDS with the control character field set to *FCFC. The TRC field does not apply and should be set to N. The original length field is Y, and the 5A field is Y.

Original length	Trimmed length	Line number	DBCS flag	RSRVD	5A	AFPDS data

- Printer device type of *AFPDS with the control character field set to *FCFC. The TRC field does not apply and should be set to N. The original length field is Y, and the 5A attribute is N.

Original length	Trimmed length	Line number	DBCS flag	RSRVD	AFPDS data

- Printer device type of *AFPDS with the control character field set to *FCFC. The TRC field does not apply and should be set to N. The original length field is N, and the 5A field is Y.

5A	AFPDS data

- Printer device type of *AFPDS with the control character field set to *FCFC. The TRC field does not apply and should be set to N. The original length field is N, and the 5A field is N.

AFPDS data

- Printer device type of *AFPDSLINE with the control character field set to *FCFC, the original length field set to Y, the 5A field set to Y, and the TRC field set to N.

  *AFPDSLINE spooled files must have control characters and lengths.

Original length	Trimmed length	line number	DBCS flag	RSRVD	CC	Line data
Original length	Trimmed length	Line number	DBCS flag	RSRVD	5A	AFPDS data

- Printer device type of *AFPDSLINE with the control character field set to *FCFC, the original length field set to Y, the 5A field set to Y, and the TRC field set to Y.

  *AFPDSLINE spooled files must have control characters and lengths.

Original length	Trimmed length	Line number	DBCS flag	RSRVD	CC	TRC	Line data
Original length	Trimmed length	Line number	DBCS flag	RSRVD	5A	AFPDS data	

Field Descriptions

AFPDS data. CHAR(*): A variable length field describing the structured fields that reside in the data stream. See the *Advanced Function Printing: Data Stream Reference* for a description of these structured fields. When the length fields are present, only one structured field can be contained in the APFDS data.

CC. CHAR(1): Carriage control.

All codes are in hexadecimal notation. The carriage control characters control writing, spacing, and skipping operations as the data is being formatted. It may or may not precede the line data. See the *Print Services Facility User's Programming Guide*, S544-3512, for VM or *Print Services Facility/MVS Applications Programming Guide*, S544-3084, for further information on the valid carriage control codes.

When control characters exist, they can be one of two types: American National Standard printer control characters or machine control characters.

DBCS flag. BIT(1): For DBCS files, whether this page starts in IGC mode. This should be zero for all lines except the first one on the page.

Line data. CHAR(*): A variable length character field composed of the actual characters to print.

Line number. BIN(15): The line number this page starts on. This should be zero for all lines except the first one on the page.

Original length. BIN(15): The original length of the record before trailing blanks were trimmed. This length does not include the first 8 bytes of the record.

RSRVD. BIT(7): Reserved field.

TRC. CHAR(1): A table-reference character.

All codes are in hexadecimal notation. A table-reference character selects a font to be used when printing the line. See the *Print Services Facility User's Programming Guide for VM*, S544-3512, or *Print Services Facility/MVS Applications Programming Guide*, S544-3084, for further information on the valid table reference codes and how they relate to the CHARS attributes and page definitions. If both CC and TRC are present, CC will be first and the TRC character will be second. The absence or presence of these characters is determined by the control character attribute and TRC attribute.

Trimmed length. BIN(15): The actual or trimmed length of the record. This length does not include the first 8 bytes of the record.

5A. CHAR(1): A special hexadecimal constant used to indicate that a structured field follows.

Restrictions for Print Data Section

Some restrictions apply as to where these record formats can appear in the buffer.

- For *LINE data records it is required that the original length, trimmed length, control character, and TRC character, if they exist, are not split across buffers.
- For *AFPDS data records it is required that the original length, trimmed length, hex 5A control character, and the next 3 bytes of data are not split across a buffer boundary.

For other data types the print text format corresponds to the architecture for that data stream.

Error Messages

| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid
| CPF33DF E Internal data area for open spooled files destroyed.
| CPF33D2 E Spooled file handle not valid.
| CPF33D3 E Value &1 not valid for buffer to read parameter.
| CPF33D4 E Value &1 not valid for end of open file parameter.
| CPF33D5 E Spooled file not opened for operation requested.
| CPF33D6 E Buffer &1 not available.
| CPF33D7 E Requested number of buffers not returned.
| CPF3330 E Necessary resource not available.
| CPF811A E User space &4 in &9 damaged.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in library &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries in library list.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9846 E Error while processing file &1 in library &2.

List Spooled Files (QUSLSPL) API

Parameters

Required Parameter Group:

1	User space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	User name	Input	Char(10)
4	Output queue and library name	Input	Char(20)
5	Form type	Input	Char(10)
6	User-specified data	Input	Char(10)

Optional Parameter:

7	Error code	I/O	Char(*)

The List Spooled Files (QUSLSPL) API is similar to the Work with Spooled Files (WRKSPLF) command. It generates a list of spooled files on the system into a user space. The list can include all spooled files or those of specific users, output queues, form types, or user-specified data values. The generated list replaces any existing lists in the user space.

Authorities and Locks

User Space Authority	*CHANGE
Library Authority	*USE
Output Queue Authority	*USE
User Space Lock	*EXCLRD

Required Parameter Group

User space and library name
INPUT; CHAR(20)
The user space that receives the generated list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The format of the spooled file information being returned. You must specify the following:

SPLF0100 Spooled file information

For more information about the SPLF0100 format, see "SPLF0100 Format" on page 26-25.

User name
INPUT; CHAR(10)
The name of the user whose files are included in the list. The possible special values are:

***ALL** Files created by all users
***CURRENT** Files created by the current user

Output queue and library name
INPUT; CHAR(20)
The name of the output queue whose files are to be included in the list, and the library in which it is located. The first 10 characters contain the output queue name, and the second 10 characters contain the library name.

You can use this special value for the output queue name:

***ALL** Files on all output queues. When you use this value, the library name must be blanks.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Form type
INPUT; CHAR(10)
The form type whose files are included in the list. The form type is the value specified on the form type parameter of the printer file. The special values supported are:

***ALL** Files for all form types
***STD** Only files that specify the standard form type

User-specified data
INPUT; CHAR(10)
The user-specified data value whose files are to be included in the list. The special value supported is:

***ALL** Files with any user-specified data values

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

SPLF0100 Format

The spooled file list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

For details about the user area and generic header fields, see "User Space Format for List APIs" on page 2-7. For details about the remaining items, see the following subtopics. For descriptions of the fields in the list, see "Field Descriptions."

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	User name specified
38	26	CHAR(10)	Output queue name specified
48	30	CHAR(10)	Output queue library name specified
58	3A	CHAR(10)	Form type
68	44	CHAR(10)	User-specified data

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User name used
10	A	CHAR(10)	Output queue name used
20	14	CHAR(10)	Output queue library name used

List Data Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User name used
10	A	CHAR(10)	Output queue name used
20	14	CHAR(10)	Output queue library name used
30	1E	CHAR(10)	Form type
40	28	CHAR(10)	User-specified data
50	32	CHAR(16)	Internal job identifier
66	42	CHAR(16)	Internal spooled file identifier

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Appendix A, "Examples."

Field Descriptions

Form type. The type of form to load in the printer to print this file.

Format name. The format of the retrieved output.

Internal job identifier. The internal identifier for the job. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Internal spooled file identifier. The input value that other programs use to improve the performance of locating the spooled file on the system. Only the spooled file APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Output queue library name specified. The name of the output queue library specified in the call to the API.

Output queue library name used. The library where the output queue is located.

Output queue name specified. The name of the output queue specified in the call to the API.

Output queue name used. The name of the output queue in which the file is located.

User name specified. The name of the user specified in the call to the API.

User name used. The name of the user that created the spooled file.

User space library name. The library containing the user space.

User space name. The name of the user space.

User-specified data. The 10 characters of user-specified data that describe the file.

Error Messages

| CPD3C21 D Format name &1 is not valid.
| CPD3C30 D Library name &1 is not valid for output queue *ALL.
| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3CF2 E Error(s) occurred during running of &1 API.
| CPF3C20 E Error found by program &1.
| CPF3C21 E Format name &1 is not valid.
| CPF3C30 E Library name &1 is not valid for output queue *ALL.
| CPF3C36 E Number of parameters, &1, entered for this API was not valid.
| CPF811A E User space &4 in &9 damaged.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9804 E Object &2 in library &3 damaged.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9838 E User profile storage limit exceeded.

Open Spooled File (QSPOPNSP) API

Parameters

Required Parameter Group:

1	Spooled file handle returned	Output	Binary(4)
2	Qualified job name	Input	Char(26)
3	Internal job identifier	Input	Char(16)
4	Internal spooled file identifier	Input	Char(16)
5	Spooled file name	Input	Char(10)
6	Spooled file number	Input	Binary(4)
7	Number of buffers to get	Input	Binary(4)
8	Error code	I/O	Char(*)

The Open Spooled File (QSPOPNSP) API opens an existing spooled file. After the existing spooled file is opened, the Get Spooled File Data (QSPGETSP) API can then get the data from the file.

Authorities and Locks

Authorities and locks are needed to use the QSPOPNSP API. Following are the details for authorities and locks.

Authorities: The requester is authorized to the spooled file if one or more of the following conditions are met.

- The requester is the owner of the spooled file.
- The requester has *READ authority to the queue on which the spooled file resides, and the queue is specified as DSPDTA(*YES).
- The requester has *SPLCTL authority.
- The requester has *JOBCTL authority, and the queue on which the spooled file resides is specified as OPRCTL(*YES) and DSPDTA(*YES or *NO).
- The requester has ownership of the queue on which the spooled file resides, and the queue is specified as AUTCHK(*OWNER) and DSPDTA(*YES or *NO).
- The requester has read, add, and delete authorities to the queue on which the spooled file resides, and the queue is specified as AUTCHK(*DTAAUT) and DSPDTA(*YES or *NO).

Locks

Spooled File Lock *LSRD

Required Parameter Group

Spooled file handle returned
OUTPUT; BINARY(4)
The handle used on subsequent get (QSPGETSP API) and close (QSPCLOSP API) operations to identify the spooled file.

Qualified job name
INPUT; CHAR(26)
The job that created the spooled file. The qualified job name has three parts:

job name
CHAR(10). A specific job name, or one of the following special values:
- * Only the job that this program is running. The rest of the job name parameter must be blank.
- *INT The internal spooled file identifier used to locate the spooled file. The user name and job number must be set to blank.

user name
CHAR(10). A specific user profile name, or blanks when the job name is * or *INT.

job number
CHAR(6). A specific job number, or blanks when the job name is * or *INT.

Internal job identifier
INPUT; CHAR(16)
The internal job identifier for the job that created the spooled file whose attributes are being retrieved. Use the Retrieve Spooled File Attributes (QUSRSPLA) API or one of these APIs to make the identifier available:

"List Spooled Files (QUSLSPL) API" on page 26-24
"List Job (QUSLJOB) API" on page 37-4
"Retrieve Job Information (QUSRJOBI) API" on page 37-15

Internal spooled file identifier
INPUT; CHAR(16)
The internal spooled file identifier for the spooled file whose attributes are being retrieved. To make the identifier available, use the QUSRSPLA API or see "List Spooled Files (QUSLSPL) API" on page 26-24.

Spooled file name
INPUT; CHAR(10)
The name of the spooled file for which you may want to retrieve the attributes. You can use this special value for the name:

***INT** The internal spooled file identifier is used to locate the spooled file.

Spooled file number
INPUT; BINARY(4)
The unique number of the spooled file. The valid range is 1 through 9999. The following special values are supported for this parameter:

0 Only one spooled file from the job has the specified file name, so the number of the spooled file is not necessary.
-1 This uses the highest-numbered spooled file with the specified file name.

Number of buffers to get
INPUT;BINARY(4)
How many buffers to get on each call to the QSPGETSP API. Valid numbers include 1 through 8, 16, 24, and 32. Any values greater than 32 must be a multiple of 32. A value greater than 1 should show some performance improvement. A value of 8 may give the best performance when using the QSPGETSP API. The following special value is supported for this parameter:

-1 Reads the entire spooled file.

If the user space cannot accommodate all buffers requested, as many complete buffers as possible will be returned in the available space. The information complete indicator in the header section of the data returned is then set to P.

If the end of an open spooled file is encountered before the buffer count is reached, the end of open spooled file parameter on the Get Spooled File Data (QSPGETSP) API determines the action taken.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

How to Select a Spooled File to Be Opened

This table illustrates the valid parameter combinations of qualified job name, internal job identifier, internal spooled file identifier, and spooled file name. The combinations of these parameters identify the spooled file to be opened. For example, when the qualified job name parameter value is *, the internal job identifier must be blank, the internal spooled file identifier must be blank, and the actual name of the spooled file must be given.

Qualified Job Name			Internal Job Identifier	Internal Spooled File Identifier	Spooled File Name
Job Name	User Name	Job Number			
Name	Name	Number	Blanks	Blanks	Name
Name	Blanks	Number	Blanks	Blanks	Name
Name	Name	Blanks	Blanks	Blanks	Name
*	Blanks	Blanks	Blanks	Blanks	Name
*INT	Blanks	Blanks	Internal job identifier	Blanks	Name
*INT	Blanks	Blanks	Internal job identifier	Internal spooled file identifier	*INT

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C33 E Spooled file number &1 is not valid.
CPF3C40 E Spooled file &4 not found.
CPF3C41 E More that one spooled file with same name.
CPF3C42 E User name or job number is not blank.
CPF3C43 E Internal job identifier not valid.
CPF3C44 E Internal spooled file identifier is not valid.
CPF3C58 E Job name specified not valid.
CPF3C59 E Internal identifier is not blanks and job name is not *INT.
CPF33C9 E Spooled file name parameter cannot be blank.
CPF33DA E Value &1 not valid for number of buffers to read parameter.
CPF33DB E Qualified job name parameter not valid with internal spooled file name.
CPF33DC E Open or create not valid for diskette files.
CPF33DD E Maximum number of open spooled files exceeded for this job.
CPF33DE E Size of internal data for opened spooled file exceeds maximum.
CPF33DF E Internal data area for opened spooled files destroyed.
CPF3309 E No files named &1 are active.
CPF3330 E Necessary resources not available.
CPF3342 E Job &5/&4/&3 not found.
CPF3343 E Duplicate job names found.
CPF3344 E File &1 number &2 no longer in the system.
CPF3492 E Not authorized to spooled file.
CPF9838 E User profile storage limit exceeded.

Put Spooled File Data (QSPPUTSP) API

Parameters

Required Parameter Group:

1	Spooled file handle returned	Input	Binary(4)
2	User space name and library name	Input	Char(20)
3	Error code	I/O	Char(*)

The Put Spooled File Data (QSPPUTSP) API puts data into a spooled file, which was created using the Create Spooled File (QSPCRTSP) API. The data put in the spooled file is taken from a user space. The data in the user space can be created using the Get Spooled File Data (QSPGETSP) API or can be created by a user application.

Before a buffer is put in a spooled file, a limited validity check is performed on the information in the user space for that buffer. The possible errors that result from the values of the fields in the user space can be classified as follows:

- The value with an error can be substituted by the default value. A CPIxxxx message is issued informing the user of the substitution.
- The value with an error causes a CPFxxxx message to be issued. The message is not issued until the validation of the information is complete or a severe error is encountered and validation cannot continue. This provides the caller with all informational messages as well as the first CPFxxxx message encountered.
- The value with an error causes a CPFxxxx message to be issued immediately.

The buffers must be in the format returned by the Get Spooled File Data (QSPGETSP) API.

Authorities and Locks

User Space Authority *CHANGE
Library Authority *USE
User Space Lock *EXCLRD

Required Parameter Group

Spooled file handle returned
INPUT; BINARY(4)
The handle returned by the QSPCRTSP API.

User space name and library name
INPUT; CHAR(20)
The name of the user space that contains the buffer of spooled information. The first 10 characters contain the user space name, and the second 10 characters contains the name of the library in which the user space is located. The special values allowed for the library name are *LIBL and *CURLIB. Both entries are left-justified. If no library is specified as the current library of the job, QGPL is used. The format of the user space is the same as that returned by QSPGETSP API. The format specified must be SPFR0200.

To see the format of the user space and how the offset values are calculated, see "Format of the User Space" on page 26-19.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Considerations When Changing or Creating a User Space

When creating your own spooled files or altering the buffers returned by the QSPGETSP API, incorrect data can be put with the QSPPUTSP API. Although errors are caught by the QSPPUTSP API, not all errors are detected.

If the order of the pages, the number of pages, or the number of lines is incorrect, problems can occur when repositioning the spooled file to print it. Repositioning means trying to start printing at a specific page other than page 1. The problems can be caused by:

- Using the restart printing function
- Changing the starting and ending print pages
- Working with an inquiry error message that allows a user to specify what page of the spooled file to restart printing

There are fields in the buffer section of the user space that, if they contain incorrect values, could cause other functions, such as displaying or copying of spooled files, to work incorrectly. These fields are in the general information section and in the page data section.

Fields in the General Information Section

If the state and IPDS data fields contain incorrect or changed values, the Display Spooled File (DSPSPLF) and Copy Spooled File (CPYSPLF) commands can be affected in the following ways:

- State field

  When this field is set to *HOMETRANS or *PAGETRANS, it indicates that the spooled file contains only IPDS transparent data. This field only affects spooled files with a device type of *IPDS.

  A changed or incorrect state field value causes the DSPSPLF and CPYSPLF commands to work the following way:

  - DSPSPLF command

    IPDS transparent data cannot be displayed by the DSPSPLF command. As a result, when the created spooled file is displayed, the data in the buffer with the state field set to *HOMETRANS or *PAGETRANS is not displayed. Furthermore, message CPI3438 (`Intelligent Printer Data Stream (IPDS) data not displayed`) appears. If the spooled file is made up entirely of buffers with the state field set to *HOMETRANS or *PAGETRANS, the spooled file is not displayed.

Message CPF3429 (File cannot be displayed or copied) is displayed.

- CPYSPLF command

  IPDS transparent data cannot be copied by the CPYSPLF command. If some buffers of a spooled file have a state field set to *HOMETRANS or *PAGETRANS, those buffers are not copied to the database member. If the spooled file is made up entirely of buffers with the state field set to *HOMETRANS or *PAGETRANS, the spooled file is not copied. Message CPF3429 (File cannot be displayed or copied) is displayed.

- IPDS data

  When this field is set to Y, it indicates that the buffer contains only IPDS data. This field only affects spooled files with a device type of *SCS.

  A changed or incorrect IPDS data field value causes the DSPSPLF and CPYSPLF commands to work the following way:

  - DSPSPLF command

    IPDS data cannot be displayed using the DSPSPLF command. As a result, when a created spooled file is displayed, the data in the buffer with the IPDS data field set to Y is not displayed. Furthermore, message CPI3437 (Intelligent printer data stream (IPDS) data not displayed) appears at the bottom of the last screen of the DSPSPLF command. If the spooled file is made up entirely of buffers with the IPDS field set to Y, the spooled file is not displayed. Message CPF3429 (File cannot be displayed or copied) is displayed.

  - CPYSPLF command

    IPDS data cannot be copied by using the CPYSPLF command. As a result, when a spooled file is copied into the database member, the data in the buffer with the IPDS data field set to Y is not copied. If the spooled file is made up entirely of buffers with the IPDS field set to Y, the spooled file is not copied. Message CPF3429 (File cannot be displayed or copied) is displayed.

Fields in the Page Data Section

If the text data start and page offset fields contain incorrect or changed values, the Display Spooled File (DSPSPLF) command can be affected in the following ways:

- Text data start

  The number of the first line where user data can start on the page. This count includes text only. The text data start field can have an incorrect value when the buffers of the original spooled file are put in the created spooled file in an order other than the original spooled file. This field can also have an incorrect value by simply changing its value to another value higher or lower than was returned when the data was gotten using the QSPGETSP API. This field affects all device types of spooled files.

  A changed or incorrect value for the text data start field causes the DSPSPLF command to work the following way:

  - DSPSPLF command

    When the text data start field contains an incorrect value, DSPSPLF may issue message CPF33F9 (Error occurred while displaying file X number Y) when attempting to find a particular string in the displayed spooled file. Also, the user is able to see only part of the spooled file. The amount of data the user sees depends on the order the buffers were put. For example, if the first buffer of the original spooled file is the last buffer in the created spooled file, the user only sees the pages in the first buffer of the created spooled file. If the second buffer of the original spooled file was the last buffer in the created spooled file, the user sees the pages in the first and second buffer of the created spooled file. The other buffers are there but not displayed.

- Page offset

  The location of the start of this page. The offset value is from the beginning of the print data. This field affects all device types of spooled files.

  A changed or incorrect value for the page offset field causes the DSPSPLF command to work the following way:

  - DSPSPLF command

    When the page offset field contains an incorrect value, DSPSPLF may issue informational message CPI3431 (Line number adjusted). The page information of the individual pages overlaps. The user sees only part of the spooled file because some pages overlap.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF33DF E Internal data area for opened spooled files destroyed.
CPF33D2 E Spooled file handle not valid.
CPF33D5 E Spooled file not opened for operation requested.
CPF33F2 E New page expected at beginning of buffer &1.
CPF33F3 E Data in buffer &1 exceeds spooled file buffer size.
CPF33F4 E Beyond end of user space &1 in library &2.
CPF33F6 E Value in generic header of user space &4 in library &5 not valid.
CPF33F7 E Value in buffer &6 of user space &4 in library &5 not valid.
CPF811A E User space &4 in &9 damaged.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in library &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.

CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to user library &1.
CPF9830 E	Cannot assign library &1.
CPF9846 E	Error while processing file &1 in library &2.
CPI33F7 I	Value in buffer &6 of user space &4 in library &5 not valid.

Retrieve Output Queue Information (QSPROUTQ) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Output queue name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Output Queue Information (QSPROUTQ) API returns information about the parameters used to create the queue, the current status of the queue, and the number of entries on the queue.

Authorities and Locks

Output queue library authority
The caller needs *READ authority to the output queue library.

Output queue authority
The caller needs one of the following:

- *READ authority to the output queue.
- Job control special authority (*JOBCTL) if the output queue is operator controlled (*OPRCTL(*YES)).
- Spool control special authority (*SPLCTL).

Output queue lock
This API needs an *EXCLRD lock on the output queue.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that receives the information requested.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable provided by the receiver variable parameter. The amount of data returned is truncated if the receiver variable is too small. A length of less than 8 is not valid.

Format name
INPUT; CHAR(8)
The content and format of the queue information being returned. The OUTQ0100 format must be used for the output queue information. See "OUTQ0100 Format" to view the information returned for this format.

Output queue name
INPUT; CHAR(20)
The name of the output queue for which information is returned. The first 10 characters contain the queue name and the second 10 characters contain the name of the library in which the queue resides.

The following special values are supported for the library name:

***LIBL** The library list used to locate the queue.
***CURLIB** The current library for the job is used to locate the queue.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

OUTQ0100 Format

The following table shows the information returned for the OUTQ0100 format. For more details about the fields in the following table see, "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Output queue name
18	12	CHAR(10)	Output queue library name
28	1C	CHAR(10)	Order of files on queue
38	26	CHAR(10)	Display any file
48	30	BINARY(4)	Job separators
52	34	CHAR(10)	Operator controlled
62	3E	CHAR(10)	Data queue name
72	48	CHAR(10)	Data queue library name
82	52	CHAR(10)	Authority to check
92	5C	BINARY(4)	Number of files
96	60	CHAR(10)	Output queue status
106	6A	CHAR(10)	Writer job name
116	74	CHAR(10)	Writer job user name
126	7E	CHAR(6)	Writer job number
136	88	CHAR(10)	Writer job status
146	92	CHAR(10)	Printer device name
156	9C	CHAR(50)	Text description

Field Descriptions

Authority to check. Indicates what type of authorities to the output queue allow the user to control all the files on the queue. The possible values are:

| *OWNER Only the owner of the output queue can control all the output files on the queue.
| *DTAAUT Any user with *READ, *ADD, or *DELETE authority to the output queue can control all output files on the queue.

| **Bytes available**. Total format data length.

| **Bytes returned**. Length of the data returned.

| **Data queue library name**. The name of the library that contains the data queue.

| **Data queue name**. Name of the data queue associated with this output queue. Whenever a spooled file goes into ready status on the output queue, an entry is placed on the data queue.

| *NONE No data queue is associated with this output queue.

| **Display any file**. Whether users who have authority to read this output queue can display the output data of any output file on this queue or only the data in their own files.

| *YES Any user having authority to read the queue can display, copy, or send the data of any file on the queue.
| *NO Users authorized to use the queue can display, copy, or send the output data of their own files only, unless they have some special authority.
| *OWNER Only the owner of a file or a user with *SPLCTL authority can display, copy, send, or move their own spooled files to another output queue.

| **Job separators**. The number of job separators to be placed at the beginning of the output for each job having spooled file entries on this output queue.

| 0-9 The desired number of job separators.
| -2 No job separators are used; instead a message is sent to the writer's message queue at the end of each job indicating that the output can be removed.

| **Number of files**. The number of spooled files that exist on the output queue.

| **Operator controlled**. Whether users with job control authority are allowed to manage or control the files on this queue. Users have job control authority if SPCAUT(*JOBCTL) is specified in their user profile. The possible values are:

| *YES Users with job control authority can control the queue and make changes to the files on the queue.
| *NO This queue and its entries cannot be controlled or changed by users with job control authority unless they also have some other special authority.

| **Order of files on queue**. The order of the spooled files on the output queue. The possible values are:

| *FIFO The queue is first-in first-out for each file. That is, on the queue, new spooled files are placed after all other spooled files that have the same priority.
| *JOBNBR The queue entries for the spooled files are sorted in priority sequence using the job number (the date and time that the job entered the system) of the job that created the spooled file.

| **Output queue library name**. The name of the library that contains the output queue.

| **Output queue name**. The name of the output queue.

| **Output queue status**. The status of the output queue. The status may be one of the following values:

| RELEASED The queue is released.
| HELD The queue is held.

| **Printer device name**. The name of the printer device. If a writer is not started to this queue, this field is blank.

| **Text description**. Text that briefly describes the output queue.

| *BLANK There is no text description of the job queue.

| **Writer job name**. The name of the writer job. If a writer job is not started to this queue, this field is blank.

| **Writer job number**. The job number associated with the writer job. If a writer job is not started to this queue, this field is blank.

| **Writer job status**. The status of the writer job. If a writer job is not started to this queue, this field is blank. The status may be one of the following values.

| STR The writer job is started to the output queue.
| END The writer job is ended.
| JOBQ The writer job is on the job queue
| HLD The writer job is held.
| MSGW The writer job is waiting for a message.

| **Writer job user name**. The name of the user who started the writer job. If a writer job is not started to this queue, this field is blank.

| Error Messages

| CPF2207 E Not authorized to use object &1 in library &3 type &2.
| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3C19 E Error occurred with receiver variable specified.
| CPF3C21 E Format name &1 is not valid.
| CPF3C24 E Length of receiver variable is not valid.
| CPF3330 E Necessary resource not available.
| CPF3357 E Output queue &1 in library &2 not found.
| CPF8122 E &8 damage on library &4.
| CPF8128 E &8 damage on output queue &4 in &9.

| Retrieve Spooled File Attributes (QUSRSPLA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified job name	Input	Char(26)
5	Internal job identifier	Input	Char(16)
6	Internal spooled file identifier	Input	Char(16)
7	Spooled file name	Input	Char(10)
8	Spooled file number	Input	Binary(4)

Optional Parameter:

9	Error code	I/O	Char(*)

The Retrieve Spooled File Attributes (QUSRSPLA) API returns specific information about a spooled file into a receiver variable. The size of the receiver variable determines the amount of information returned. You can specify the spooled file for which information is returned either with the internal job and spooled file identifiers, or with a specific job name, spooled file name, and spooled file number.

For RPG, Pascal, COBOL, and System C/400 PRPQ examples using this API, see Appendix A, "Examples."

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The receiver variable that receives the information requested. You can specify the size of the area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results are not predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the information to be returned for the specified spooled file. The valid format names are:

SPLA0100 Basic spooled file attributes. For more information about this format, see "SPLA0100 Format" on page 26-33.

SPLA0200 Detailed spooled file attributes needed to print or duplicate the spooled file. For more information about this format, see "SPLA0200 Format" on page 26-40.

Qualified job name
INPUT; CHAR(26)
The job that created the spooled file. The qualified job name has three parts:

job name CHAR(10). A specific job name, or one of the following special values:
- * Only the job that this program is running. The rest of the job name parameter must be blank.
- *INT The internal spooled file identifier used to locate the spooled file. The user name and job number must be set to blank.

user name CHAR(10). A specific user profile name, or blanks when the job name is * or *INT.

job number CHAR(6). A specific job number, or blanks when the job name is * or *INT.

Internal job identifier
INPUT; CHAR(16)
The internal job identifier for the job that created the spooled file whose attributes are to be retrieved. Use the QUSRSPLA API or one of these APIs to make the identifier available:

- "List Spooled Files (QUSLSPL) API" on page 26-24
- "List Job (QUSLJOB) API" on page 37-4
- "Retrieve Job Information (QUSRJOBI) API" on page 37-15

Internal spooled file identifier
INPUT; CHAR(16)
The internal spooled file identifier for the spooled file whose attributes are retrieved. To make the identifier available, use the QUSRSPLA API or see "List Spooled Files (QUSLSPL) API" on page 26-24.

Spooled file name
INPUT; CHAR(10)
The name of the spooled file for which you may want to retrieve the attributes. You can use this special value for the name:

*INT The internal spooled file identifier is used to locate the spooled file.

Spooled file number
INPUT; BINARY(4)
The unique number of the spooled file. The valid range is 1 through 9999. The following special values are supported for this parameter:

- 0 Only one spooled file from the job has the specified file name, so the number of the spooled file is not necessary.
- -1 This uses the highest-numbered spooled file with the specified file name.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

How to Select a Spooled File to Retrieve Its Attributes

This table illustrates the valid parameter combinations of qualified job name, internal job identifier, internal spooled file identifier, and spooled file name. The combinations of these parameters identify the spooled file from which attributes can be retrieved. For example, when the qualified job name parameter value is *, the internal job identifier must be blank, the internal spooled file identifier must be blank, and the actual name of the spooled file must be given.

Qualified Job Name			Internal Job Identifier	Internal Spooled File Identifier	Spooled File Name
Job Name	**User Name**	**Job Number**			
Name	Name	Number	Blanks	Blanks	Name
Name	Blanks	Number	Blanks	Blanks	Name
Name	Name	Blanks	Blanks	Blanks	Name
*	Blanks	Blanks	Blanks	Blanks	Name
*INT	Blanks	Blanks	Internal job identifier	Blanks	Name
*INT	Blanks	Blanks	Internal job identifier	Internal spooled file identifier	*INT

SPLA0100 Format

The following table shows the information returned for the SPLA0100 format. For a detailed description of each field, see "Field Descriptions" on page 26-35.

Offset		Type	Field
Dec	**Hex**		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(16)	Internal job identifier
24	18	CHAR(16)	Internal spooled file identifier
40	28	CHAR(10)	Job name
50	32	CHAR(10)	User name
60	3C	CHAR(6)	Job number
66	42	CHAR(10)	Spooled file name
76	4C	BINARY(4)	Spooled file number
80	50	CHAR(10)	Form type
90	5A	CHAR(10)	User-specified data
100	64	CHAR(10)	Status
110	6E	CHAR(10)	File available
120	78	CHAR(10)	Hold file before written
130	82	CHAR(10)	Save file after written
140	8C	BINARY(4)	Total pages
144	90	BINARY(4)	Page or record being written
148	94	BINARY(4)	Starting page
152	98	BINARY(4)	Ending page
156	9C	BINARY(4)	Last page printed
160	A0	BINARY(4)	Restart printing
164	A4	BINARY(4)	Total copies
168	A8	BINARY(4)	Copies left to produce
172	AC	BINARY(4)	Lines per inch
176	B0	BINARY(4)	Characters per inch
180	B4	CHAR(2)	Output priority
182	B6	CHAR(10)	Output queue name
192	C0	CHAR(10)	Output queue library name
202	CA	CHAR(7)	Date file opened
209	D1	CHAR(6)	Time file opened
215	D7	CHAR(10)	Device file name
225	E1	CHAR(10)	Device file library name
235	EB	CHAR(10)	Program that opened file name
245	F5	CHAR(10)	Program that opened file library name
255	FF	CHAR(15)	Accounting code

Offset		Type	Field
Dec	Hex		
270	10E	CHAR(30)	Print text
300	12C	BINARY(4)	Record length
304	130	BINARY(4)	Maximum records
308	134	CHAR(10)	Device type
318	13E	CHAR(10)	Printer device type
328	148	CHAR(12)	Document name
340	154	CHAR(64)	Folder name
404	194	CHAR(8)	System/36 procedure name
412	19C	CHAR(10)	Print fidelity
422	1A6	CHAR(1)	Replace unprintable characters
423	1A7	CHAR(1)	Replacement character
424	1A8	BINARY(4)	Page length
428	1AC	BINARY(4)	Page width
432	1B0	BINARY(4)	Number of separators
436	1B4	BINARY(4)	Overflow line number
440	1B8	CHAR(10)	DBCS data
450	1C2	CHAR(10)	DBCS extension characters
460	1CC	CHAR(10)	DBCS shift-out shift-in (SO/SI) spacing
470	1D6	CHAR(10)	DBCS character rotation
480	1E0	BINARY(4)	DBCS characters per inch
484	1E4	CHAR(10)	Graphic character set
494	1EE	CHAR(10)	Code page
504	1F8	CHAR(10)	Form definition name
514	202	CHAR(10)	Form definition library name
524	20C	BINARY(4)	Source drawer
528	210	CHAR(10)	Printer font
538	21A	CHAR(6)	System/36 spooled file identifier
544	220	BINARY(4)	Page rotation
548	224	BINARY(4)	Justification
552	228	CHAR(10)	Print on both sides (duplex)
562	232	CHAR(10)	Fold records
572	23C	CHAR(10)	Control character
582	246	CHAR(10)	Align forms
592	250	CHAR(10)	Print quality
602	25A	CHAR(10)	Form feed
612	264	CHAR(71)	Volumes (array)
683	2AB	CHAR(17)	File label identifier
700	2BC	CHAR(10)	Exchange type
710	2C6	CHAR(10)	Character code
720	2D0	BINARY(4)	Total records
724	2D4	BINARY(4)	Multiple up (pages per side)
728	2D8	CHAR(10)	Front overlay name
738	2E2	CHAR(10)	Front overlay library name
748	2EC	PACKED(15,5)[1]	Front overlay offset down
756	2F4	PACKED(15,5)[1]	Front overlay offset across
764	2FC	CHAR(10)	Back overlay name
774	306	CHAR(10)	Back overlay library name
784	310	PACKED(15,5)[1]	Back overlay offset down
792	318	PACKED(15,5)[1]	Back overlay offset across
800	320	CHAR(10)	Unit of measure
810	32A	CHAR(10)	Page definition name
820	334	CHAR(10)	Page definition library name
830	33E	CHAR(10)	Line spacing
840	348	PACKED(15,5)[1]	Point size
848	350	PACKED(15,5)[1]	Front margin offset down
856	358	PACKED(15,5)[1]	Front margin offset across
864	360	PACKED(15,5)[1]	Back margin offset down
872	368	PACKED(15,5)[1]	Back margin offset across
880	370	PACKED(15,5)[1]	Length of page
888	378	PACKED(15,5)[1]	Width of page
896	380	CHAR(10)	Measurement method
906	38A	CHAR(1)	Advanced Function Printing (AFP) resource
907	38B	CHAR(10)	Character set name
917	395	CHAR(10)	Character set library name
927	39F	CHAR(10)	Code page name
937	3A9	CHAR(10)	Code page library name

Offset		Type	Field
Dec	Hex		
947	3B3	CHAR(10)	Coded font name
957	3BD	CHAR(10)	Coded font library name
967	3C7	CHAR(10)	DBCS-coded font name
977	3D1	CHAR(10)	DBCS-coded font library name
987	3DB	CHAR(10)	User-defined file
1 COBOL can handle only an 18-digit number. PACKED(15,5) is too big. You must use a 1-character FILLER field followed by a PACKED(13,5) field instead.			

Field Descriptions

Accounting code. An identifier assigned by the system to record the resources used to write this file.

Advanced Function Printing (AFP) resource. Whether this spooled file uses AFP resources external to this spooled file, for example, page segments or overlays. The possible values are:

Y The spooled file refers to AFP resources external to the spooled file.
N The spooled file does not refer to AFP resources external to the spooled file.

Align forms. Whether a forms alignment message is sent prior to printing this file. The options are *YES or *NO.

Back margin offset across. For the back side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are:

0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Back margin offset down. For the back side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are:

-1 The back margin offsets are the same as the front margin offsets.
-2 *DEVD. For printers configured AFP(*YES), no print border is used for back margin offsets across and down. Otherwise, the offset values are 0.
0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Back overlay library name. The name of the library containing the back overlay. The possible values are:

**CURLIB* The current library for the job locates the back overlay.
**LIBL* The library list locates the back overlay.
library name This library is searched for the back overlay.

Back overlay name. The name of the back overlay (the material that prints on the back side of each page). The possible values are:

**FRONTOVL* The back overlay is the same as the front overlay.
**NONE* The file does not use a back overlay.
back overlay name The name of the back overlay.

Back overlay offset across. The offset across from the point of origin where the overlay is printed. The possible values are:

0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Back overlay offset down. The offset down from the point of origin where the overlay is printed. The possible values are:

0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Bytes available. The amount of data available to the calling program. In other words, the receiver variable could get this much data from this format if the receiver variable were this large or larger.

Bytes returned. The amount of data returned to the calling program in the receiver variable.

Character code. Whether the coding for the diskette file is in ASCII or EBCDIC form.

Character set library name. The name of the library containing the font character set object. The possible values are:

**LIBL* The library list is used to locate the font character set object.
library name This library is searched for the font character set object.

Character set name. The name of the font character set object used to print this file. The possible values are:

**FONT* The information specified on the font parameter is used instead of the character set and code page.
character set name The name of the font character set object to use.

Characters per inch. The number (in tenths) of characters per horizontal inch, defined in the printer file. The value 100 indicates 10 characters per inch.

Code page. The mapping of graphic characters to code points for this printer. For *DEVD, the system gets the code page from the printer device description.

Code page library name. The name of the library containing the code page used to print this spooled file. The possible values are:

**LIBL*	The library list is used to locate the code page.
library name	This library is searched for the code page name.

Code page name. The name of the code page used to print this spooled file.

Code pages are groups of characters. Within a code page, unique hexadecimal identifiers are assigned to each of the characters.

Coded font library name. The name of the library containing the coded font used to print this spooled file. The possible values are:

**LIBL*	The library list is used to locate the coded font.
library name	This library is searched for the coded font.

Coded font name. The name of the coded font used to print this spooled file. A **coded font** is an AFP resource composed of a character set and a code page. The possible values are:

**FNTCHRSET*
: The values used are the values specified on the character set name and library name and code page name and library name fields.

coded font name
: The name of the coded font used to print this spooled file.

Control character. Whether this printer file uses the American National Standard printer control character. The possible values are:

**NONE*	No print control characters are passed in the data that is printed.
**FCFC*	The first character of every record is an American National Standard printer control character.

Copies left to produce. The remaining number of copies to be produced on the printer. Valid values are 1 through 255.

Date file opened. The date that the file was opened in the CYYMMDD format.

C	Century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
YY	Year
MM	Month
DD	Day

DBCS character rotation. Whether this printer file causes the double-byte character set (DBCS) characters to be rotated 90 degrees counterclockwise before printing. The possible values are *YES and *NO.

DBCS characters per inch. The number of double-byte characters to be printed per inch. The possible values are:

-1	*CPI: One-half of the characters per inch value.
-2	*CONDENSED: 20 double-byte characters per 3 inches.
5	5 characters per inch.
6	6 characters per inch.
10	10 characters per inch.

DBCS-coded font library name. The name of the library containing the DBCS-coded font used to print this spooled file. The possible values are:

**CURLIB*	The current library is searched for the DBCS-coded font.
**LIBL*	The library list is used to locate the DBCS-coded font.
library name	This library is searched for the DBCS-coded font.

DBCS-coded font name. The name of the DBCS-coded font used to print DBCS-coded data on printers configured as AFP(*YES). The possible values are:

**SYSVAL*
: The DBCS-coded font specified in the system value is used.

DBCS-coded font name
: The name of the DBCS-coded font being used.

DBCS data. Whether the file can contain double-byte character set (DBCS) data. The options are *YES or *NO.

DBCS extension characters. Whether the system uses the extension character processing function. The possible values are:

**YES*	The system processes DBCS extension characters.
**NO*	The system does not process DBCS extension characters; it prints extension characters as the undefined character.

DBCS shift-out shift-in (SO/SI) spacing. The presentation of shift-out and shift-in characters when printed. The possible values are:

**YES*	Shift-out and shift-in characters occupy one space.
**NO*	Shift-out and shift-in characters occupy no space.
**RIGHT*	Shift-out characters occupy no space; shift-in characters occupy 2 spaces.

Device file library name. The name of the library that contains the device file.

Device file name. The name of the device file used to create the spooled file.

Device type. The type of device file for which this file is intended. The possible values are PRINTER, DISKETTE, or TAPE.

Document name. The name of the document that was the source of the spooled file.

Ending page. The page at which printing is to end for the file. 0 or 2147483647 indicates the last page. To print the entire file, set this value to 0, 2147483647, or the last page number.

Exchange type. The exchange type of the diskette file. The possible values are:

**STD* The system allows the exchange type based on the diskette type and sector size.
**BASIC* The basic exchange type.
**H* The H exchange type.
**I* The I exchange type.

File available. The time when this file becomes available to an output device for processing. The possible values are:

**IMMED* The file is available as soon as the file is opened.
**FILEEND* The file is available as soon as the file is closed.
**JOBEND* The file is available when the job that created the file is completed.

File label identifier. The diskette label used when the system last saved the object.

Fold records. Whether records exceeding the printer forms width are folded (wrapped) to the next line. The possible values are *YES or *NO.

Folder name. The name of the folder that contains the source document.

Form definition library name. The name of the library that contains the form definition.

Form definition name. The name of the form definition to use for this print request.

Form feed. The manner in which forms feed to the printer. The possible values are:

**CONT* Continuous forms
**CUT* Manually fed cut forms
**AUTOCUT* Automatically fed cut forms
**DEVD* As defined in the device description

Form type. The type of form to be loaded in the printer to print this file.

Front margin offset across. For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are:

0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Front margin offset down. For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are:

-2 *DEVD. For printers configured AFP(*YES), no print border is used for front margin offsets across and down. Otherwise, the offset values are 0.
0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Front overlay library name. The name of the library containing the front overlay. The possible values are:

**CURLIB* The current library for the job locates the front overlay.
**LIBL* The library list locates the front overlay.
library name This library is searched for the front overlay.

Front overlay name. The name of the front overlay (the material that prints on the front side of each page). The possible values are:

**NONE* The file does not use the front overlay.
front overlay name
The name of the front overlay.

Front overlay offset across. The offset across from the point of origin where the overlay is printed. The possible values are:

0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Front overlay offset down. The offset down from the point of origin where the overlay is printed. The possible values are:

0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Graphic character set. The set of graphic characters to be used when printing this file. For *DEVD, the system gets the graphic character set from the printer device description.

Hold file before written. Whether the file is held. The hold parameter handles this function on a Create Printer File (CRTPRTF), Override Printer File (OVRPRTF), or Change Printer File (CHGPRTF) command. The possible values are:

| *YES The file is held.
| *NO The file is not held.

| **Internal job identifier**. The internal identifier for the job. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

| **Internal spooled file identifier**. The input value used by other programs to improve the performance of locating the spooled file on the system. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

| **Job name**. The name of the job that created the spooled file.

| **Job number**. The number of the job that created the spooled file.

| **Justification**. The percentage that the output is right-justified. The possible values are 100, 50, or 0.

| **Last page printed**. The number of the last printed page in the file if printing ended before the job completed processing.

| **Length of page**. The length of a page. Units of measurement are specified in the measurement method field.

| **Line spacing**. How a file's line data records are spaced when printed. This information is returned only for *LINE and *AFPDSLINE printer device type files. The possible values are:

| *SINGLE Single-spaced
| *DOUBLE Double-spaced
| *TRIPLE Triple-spaced
| *CTLCHAR The control character field determines line spacing. The control character field can have one of these values:
| *NONE No print control characters are passed in the data that is printed.
| *FCFC The first character of every record is an American National Standard printer control character.

| **Lines per inch**. The number (in tenths) of lines per vertical inch defined in the printer file. A value of 40 indicates 4 lines per inch.

| **Maximum records**. The maximum number of records allowed in the file at the time the file was opened. A value of 0 indicates no maximum.

| **Measurement method**. The measurement method used for the length of page and width of page fields. The possible values are:

| *ROWCOL Uses rows and columns as the unit of measure.
| *UOM Uses the value specified on the unit of measurement parameter (UOM). UOM is either inches (*INCH) or centimeters (*CM).

| **Multiple up (pages per side)**. The number of logical pages that print on each side of each physical page when this file is printed. The possible values are 1, 2, and 4.

| **Number of separators**. The number of file separator pages placed at the beginning of each copy of this file.

| **Output priority**. The priority of the spooled file. The priority ranges from 1 (highest) to 9 (lowest).

| **Output queue library name**. The name of the library that contains the output queue.

| **Output queue name**. The name of the output queue where the file is located.

| **Overflow line number**. The last line to be printed before the data being printed overflows to the next page.

| **Page definition library name**. The name of the library in which the page definition resides. This information is returned only for *LINE or *AFPDSLINE printer device type files.

| **Page definition name**. The name of the page definition to use for the file. This information is returned only for *LINE or *AFPDSLINE printer device type files.

| **Page length**. The page length (in lines per page) used by the spooled file. The valid range is row 1 through 255. The value used should not exceed the actual length of the page used.

| **Page or record being written**. The page number or record number currently being written. The page number may be lower or higher than the page number actually being printed because of buffering done by the system. The page number shown may be zero if:

- The printer file is routed to a diskette unit.
- The writer is currently printing job or file separators for the file.

| **Page rotation**. The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. The possible values are:

| -1 *AUTO: Computer output reduction is done automatically if the output is too large to fit on the form, regardless of the print quality.
| -2 *DEVD: Page rotation is obtained from the printer device description.
| -3 *COR: Output created for a form 13.2 inches wide by 11.0 inches long is adjusted to print on a form 11.0 inches wide by 8.5 inches long.
| 0 No rotation is done. Printing starts at the edge loaded into the printer first, and is parallel to that edge.
| 90 Text is rotated 90 degrees clockwise from the 0-degree writing position.

180	Text is rotated 180 degrees clockwise from the 0-degree writing position.
270	Text is rotated 270 degrees clockwise from the 0-degree writing position.

Page width. The page width (in characters per printed line) used by the spooled file. The valid range is column 1 through 378, although some printers have a page width less than 378. The value should not exceed the actual width of the page used.

Point size. The point size in which this file's characters should be printed.

Print fidelity. The kind of error handling that is performed when printing. The possible values are:

**ABSOLUTE*	The file is printed only if it can be printed exactly as specified in the data stream.
**CONTENT*	The printing overrides errors in the data stream and continues printing with the printer's best quality based on the content fidelity.

Print on both sides (duplex). How the information prints. The possible values are:

**FORMDF*	The file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, and *AFPDSLINE printer device type files.
**NO*	The printing on the page is on one side only.
**YES*	The printing is on both sides of the page with the top of each page the same for both sides.
**TUMBLE*	The printing is on both sides with the top of one printed page at the opposite end from the top of the other printed page.

Print quality. The print quality that is used when printing this spooled file. The possible values are:

**STD*	Standard
**DRAFT*	Draft
**NLQ*	Near-letter quality
**FASTDRAFT*	Prints at a faster speed than *DRAFT

Print text. The text that is printed at the bottom of each page of printed output and on separator pages.

Printer device type. The type of data stream used to represent the file. The possible values are:

**AFPDS*	Advanced Function Printing data stream
**AFPDSLINE*	AFPDS data mixed with 1403 line data
**IPDS*	Intelligent printer data stream
**LINE*	1403 line data
**SCS*	Systems Network Architecture (SNA) character stream
**USERASCII*	ASCII data

Printer font. The printer font used. If *DEVD is shown, the printer uses the font defined in the printer device description. If *CPI is shown, the file is printed with a font that has the pitch specified by the CPI (character per inch) field.

Program that opened file library name. The name of the library that contains the program that opened the file.

Program that opened file name. The name of the program that opened the spooled file.

Record length. The length of the file's records. If the field shows -1 (the special value for *RCDFMT), this is an externally defined file and the length is included in the file definition. The length includes the extra length of 1 for carriage controls if it exists.

Replace unprintable characters. Whether characters that cannot be printed are to be replaced with another character. The options are Y (yes) or N (no).

Replacement character. The character that replaces any unprintable characters. This field has a value if the replace unprintables field specifies Y.

Restart printing. The number of the page where printing restarts. When you specify a value while the file is printing, the writer stops printing and restarts on the specified page. If the file is not currently printing, this change takes effect after the first copy prints. The possible values are:

-1	*STRPAGE: The starting page specified in the PAGERANGE parameter is the page on which to restart printing.
-2	*ENDPAGE: Only the last page prints.
-3	*NEXT: The next page of the file to print is the page where printing is restarted.
restart page	The page number you specify is where printing restarts.

Save file after written. Whether this file is to be saved after it is written. The possible values are *YES and *NO.

Source drawer. The drawer to be used when the automatically cut form feed option is selected. The possible values are:

1-255	The drawer number.
-1	*E1, the envelope drawer.
-2	*FORMDF, indicating that the file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, or *AFPDSLINE printer device type files.

Spooled file name. The name of the spooled file whose information is retrieved.

Spooled file number. The spooled file number of the specified file.

Starting page. The page at which printing is to start for the file. The possible values are:

0	Printing starts on page 1.
-1	The ending page value.
1	The entire file prints.

Status. The status of the file is one of the following values:

**CLOSED*	The program completely processed the file, but SCHEDULE(*JOBEND) was specified, and the job that produced the file has not yet finished.
**HELD*	The file is held.
**MESSAGE*	The file has a message requiring a reply or an action.
**OPEN*	The file was not completely processed and is not ready for a writer.
**PENDING*	The file is pending to be printed.
**PRINTER*	The complete file was sent to the printer, but a print complete status was not sent back.
**READY*	The file is available to be written.
**SAVED*	The file was written and is saved. This file remains saved until it is released.
**WRITING*	The writer is writing the file on a diskette or a printer device.

System/36 procedure name. The name of the procedure running when the spooled file was created.

System/36 spooled file identifier. The 6-character identifier assigned to the spooled file.

Time file opened. The time that the file was opened in the HHMMSS format where:

HH	Hour
MM	Minutes
SS	Seconds

Total copies. The total number of copies to be produced for this printer file.

Total pages. The number of pages this printer file contains.

Total records. If the spooled file data stream is *AFPDS, *AFPDSLINE, or *LINE or if the spooled file device type is diskette, this field contains the total number of records in the printer file or the diskette file. The field is blank if the file is open.

Unit of measure. The unit of measure to use for specifying distances. The unit of measure is used with certain parameters of the printer file. Page definitions and form definitions are examples of these parameters. The possible values are:

**CM*	Centimeters
**INCH*	Inches

User name. The name of the user that created the spooled file.

User-defined file. Whether the spooled file was created using an API. The possible values are:

**YES*	The spooled file was created using the Create Spooled File (QSPCRTSP) API.
**NO*	The spooled file was created by normal system functions.

User-specified data. The 10 characters of user-specified data that describe the file.

Volumes (array). The diskette volumes used for saving the object. The variable returns a maximum of 10 six-character volumes. The volume IDs begin in character positions 1, 8, 15, 22, 29, 36, 43, 50, 57, and 64. If more than 10 volumes are used, a 1 is returned in the 71st character of the variable; otherwise, the 71st character is blank. The variable is returned blank if the object was saved to a save file, or if it was never saved.

Width of page. The width of a page. Units of measurement are specified in the measurement method field.

SPLA0200 Format

The following table shows the information returned for the SPLA0200 format. For more details about the fields in the following table, see "Field Descriptions" on page 26-43.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(8)	Format name
16	10	CHAR(16)	Internal job identifier
32	20	CHAR(16)	Internal spooled file identifier
48	30	CHAR(10)	Job name
58	3A	CHAR(10)	User name
68	44	CHAR(6)	Job number
74	4A	CHAR(10)	Spooled file name
84	54	BINARY(4)	Spooled file number
88	58	CHAR(10)	Form type
98	62	CHAR(10)	User-specified data
108	6C	CHAR(10)	Status
118	76	CHAR(10)	File available
128	80	CHAR(10)	Hold file before written
138	8A	CHAR(10)	Save file after written
148	94	BINARY(4)	Total pages
152	98	BINARY(4)	Page or record being written
156	9C	BINARY(4)	Starting page
160	A0	BINARY(4)	Ending page
164	A4	BINARY(4)	Last page printed
168	A8	BINARY(4)	Restart printing
172	AC	BINARY(4)	Total copies
176	B0	BINARY(4)	Copies left to produce

Offset		Type	Field
Dec	Hex		
180	B4	BINARY(4)	Lines per inch
184	B8	BINARY(4)	Characters per inch
188	BC	CHAR(2)	Output priority
190	BE	CHAR(10)	Output queue name
200	C8	CHAR(10)	Output queue library name
210	D2	CHAR(7)	Date file opened
217	D9	CHAR(6)	Time file opened
223	DF	CHAR(10)	Device file name
233	E9	CHAR(10)	Device file library name
243	F3	CHAR(10)	Program that opened file name
253	FD	CHAR(10)	Program that opened file library name
263	107	CHAR(15)	Accounting code
278	116	CHAR(30)	Print text
308	134	BINARY(4)	Record length
312	138	BINARY(4)	Maximum records
316	13C	CHAR(10)	Device type
326	146	CHAR(10)	Printer device type
336	150	CHAR(12)	Document name
348	15C	CHAR(64)	Folder name
412	19C	CHAR(8)	System/36 procedure name
420	1A4	CHAR(10)	Print fidelity
430	1AE	CHAR(1)	Replace unprintable characters
431	1AF	CHAR(1)	Replacement character
432	1B0	BINARY(4)	Page length
436	1B4	BINARY(4)	Page width
440	1B8	BINARY(4)	Number of separators
444	1BC	BINARY(4)	Overflow line number
448	1C0	CHAR(10)	DBCS data
458	1CA	CHAR(10)	DBCS extension characters
468	1D4	CHAR(10)	DBCS shift-out shift-in (SO/SI) spacing
478	1DE	CHAR(10)	DBCS character rotation
488	1E8	BINARY(4)	DBCS characters per inch
492	1EC	CHAR(10)	Graphic character set

Offset		Type	Field
Dec	Hex		
502	1F6	CHAR(10)	Code page
512	200	CHAR(10)	Form definition name
522	20A	CHAR(10)	Form definition library name
532	214	BINARY(4)	Source drawer
536	218	CHAR(10)	Printer font
546	222	CHAR(6)	System/36 spooled file identifier
552	228	BINARY(4)	Page rotation
556	22C	BINARY(4)	Justification
560	230	CHAR(10)	Print on both sides (duplex)
570	23A	CHAR(10)	Fold records
580	244	CHAR(10)	Control character
590	24E	CHAR(10)	Align forms
600	258	CHAR(10)	Print quality
610	262	CHAR(10)	Form feed
620	26C	CHAR(71)	Volumes (array)
691	2B3	CHAR(17)	File label identifier
708	2C4	CHAR(10)	Exchange type
718	2CE	CHAR(10)	Character code
728	2D8	BINARY(4)	Total records
732	2DC	BINARY(4)	Multiple up (pages per side)
736	2E0	CHAR(10)	Front overlay name
746	2EA	CHAR(10)	Front overlay library name
756	2F4	PACKED(15,5)[1]	Front overlay offset down
764	2FC	PACKED(15,5)[1]	Front overlay offset across
772	304	CHAR(10)	Back overlay name
782	30E	CHAR(10)	Back overlay library name
792	318	PACKED(15,5)[1]	Back overlay offset down
800	320	PACKED(15,5)[1]	Back overlay offset across
808	328	CHAR(10)	Unit of measure
818	332	CHAR(10)	Page definition name
828	33C	CHAR(10)	Page definition library name
838	346	CHAR(10)	Line spacing
848	350	PACKED(15,5)[1]	Point size
856	358	BINARY(4)	Maximum spooled data record size

Offset		Type	Field
Dec	Hex		
860	35C	BINARY(4)	Spooled file buffer size
864	360	CHAR(6)	Spooled file level
870	366	ARRAY(4)	Coded font array
886	376	CHAR(10)	Channel mode
896	380	ARRAY(12)	Channel value array
944	3B0	CHAR(8)	Graphics token
952	3B8	CHAR(10)	Record format
962	3C2	CHAR(2)	Reserved
964	3C4	PACKED(15,5)[1]	Height of drawer 1
972	3CC	PACKED(15,5)[1]	Width of drawer 1
980	3D4	PACKED(15,5)[1]	Height of drawer 2
988	3DC	PACKED(15,5)[1]	Width of drawer 2
996	3E4	BINARY(4)	Number of buffers
1000	3E8	BINARY(4)	Maximum forms width
1004	3EC	BINARY(4)	Alternate forms width
1008	3F0	BINARY(4)	Alternate forms length
1012	3F4	BINARY(4)	Alternate lines per inch
1016	3F8	CHAR(2)	System/38 Text Utility flags
1018	3FA	CHAR(1)	File open
1019	3FB	CHAR(1)	Page count estimated
1020	3FC	CHAR(1)	File stopped on page boundary
1021	3FD	CHAR(1)	TRC for 1403
1022	3FE	CHAR(1)	Define characters
1023	3FF	CHAR(1)	Characters per inch changes
1024	400	CHAR(1)	Transparency
1025	401	CHAR(1)	Double-wide characters
1026	402	CHAR(1)	DBCS character rotation commands
1027	403	CHAR(1)	Extended code page
1028	404	CHAR(1)	FFT emphasis
1029	405	CHAR(1)	3812 SCS
1030	406	CHAR(1)	Set Line Density command
1031	407	CHAR(1)	Graphics error actions
1032	408	CHAR(1)	5219 commands
1033	409	CHAR(1)	3812 SCS commands

Offset		Type	Field
Dec	Hex		
1034	40A	CHAR(1)	Field outlining
1035	40B	CHAR(1)	Final form text
1036	40C	CHAR(1)	Bar code
1037	40D	CHAR(1)	Color
1038	40E	CHAR(1)	Drawer change
1039	40F	CHAR(1)	Character ID
1040	410	CHAR(1)	Lines per inch changes
1041	411	CHAR(1)	Font
1042	412	CHAR(1)	Highlight
1043	413	CHAR(1)	Page rotate
1044	414	CHAR(1)	Subscript
1045	415	CHAR(1)	Superscript
1046	416	CHAR(1)	DDS
1047	417	CHAR(1)	Final form feed
1048	418	CHAR(1)	SCS data
1049	419	CHAR(1)	User-generated data stream
1050	41A	CHAR(1)	Graphics
1051	41B	CHAR(1)	Unrecognizable data
1052	41C	CHAR(1)	ASCII transparency
1053	41D	CHAR(1)	IPDS transparent data
1054	41E	CHAR(1)	OfficeVision/400
1055	41F	CHAR(1)	Lines-per-inch (lpi) value not supported
1056	420	CHAR(1)	CPA3353 message
1057	421	CHAR(1)	Set exception
1058	422	CHAR(1)	Carriage control characters
1059	423	CHAR(1)	Page position
1060	424	CHAR(1)	Character not valid
1061	425	CHAR(1)	Lengths present
1062	426	CHAR(1)	5A present
1063	427	CHAR(1)	Reserved
1064	428	BINARY(4)	Number of font array entries
1068	42C	BINARY(4)	Number of resource library entries
1072	430	CHAR(1153)	Font equivalence array
2225	8B1	CHAR(631)	Resource library array
2856	B28	CHAR(1)	AS/400-created AFPDS
2857	B29	CHAR(295)	Reserved

Offset		Type	Field
Dec	Hex		
3152	C50	PACKED(15,5)[1]	Front margin offset down
3160	C58	PACKED(15,5)[1]	Front margin offset across
3168	C60	PACKED(15,5)[1]	Back margin offset down
3176	C68	PACKED(15,5)[1]	Back margin offset across
3184	C70	PACKED(15,5)[1]	Length of page
3192	C78	PACKED(15,5)[1]	Width of page
3200	C80	CHAR(10)	Measurement method
3210	C8A	CHAR(1)	Advanced Function Printing (AFP) resource
3211	C8B	CHAR(10)	Character set name
3221	C95	CHAR(10)	Character set library name
3231	C9F	CHAR(10)	Code page name
3241	CA9	CHAR(10)	Code page library name
3251	CB3	CHAR(10)	Coded font name
3261	CBD	CHAR(10)	Coded font library name
3271	CC7	CHAR(10)	DBCS-coded font name
3281	CD1	CHAR(10)	DBCS-coded font library name
3291	CDB	CHAR(10)	User-defined file

1 COBOL can handle only an 18-digit number. PACKED(15,5) is too big. You must use a 1-character FILLER field followed by a PACKED(13,5) field instead.

Field Descriptions

Accounting code. An identifier assigned by the system to record the resources used to write this file.

Advanced Function Printing (AFP) resource. Whether this spooled file refers to AFP resources external to this spooled file, for example, page segments or overlays. The possible values are:

Y The spooled file refers to AFP resources external to the spooled file.
N The spooled file does not refer to AFP resources external to the spooled file.

Align forms. Whether a forms alignment message is sent prior to printing this file. The options are *YES or *NO.

Alternate forms length. The length of the alternate forms in lines. This field applies only to files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields.

Alternate forms width. The width of the alternate forms in character positions. This field applies only to files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields.

Alternate lines per inch. The lines per inch for the alternate forms. This field applies only to files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields.

AS/400-created AFPDS. Whether the spooled file was created on an AS/400 system using a printer file with the device type (DEVTYPE) parameter value set to *AFPDS.

ASCII transparency. For SCS files, whether ASCII commands are embedded in the ASCII transparency command. The ASCII transparency command is command_ID/command_length/ASCII_data. The command ID is a 1-byte field with value hex 03. The command length is a 1-byte field that contains the length of the command length field and the ASCII data field.

Valid values are Y (yes) or N (no).

Back margin offset across. For the back side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are:

0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-52). The maximum offset is 57.79 centimeters or 22.75 inches.

Back margin offset down. For the back side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are:

-1 *FRONTMGN. The back margin offsets are the same as the front margin offsets.
-2 *DEVD. For printers configured AFP(*YES), no print border is used for back margin offsets across and down. Otherwise, the offset values are 0.
0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-52). The maximum offset is 57.79 centimeters or 22.75 inches.

Back overlay library name. The name of the library containing the back overlay. The possible values are:

**CURLIB* The current library for the job locates the back overlay.
**LIBL* The library list locates the back overlay.
library name This library is searched for the back overlay.

Back overlay name. The name of the back overlay (the material that prints on the back side of each page). The possible values are:

**FRONTOVL*
: The back overlay is the same as the front overlay.

**NONE*
: The file does not use a back overlay.

back overlay name
: The name of the back overlay.

Back overlay offset across. The offset across from the point of origin where the overlay is printed. The possible values are:

0 – 57.79
: The offset in inches or centimeters, according to the unit of measure value (see page 26-52). The maximum offset is 57.79 centimeters or 22.75 inches.

Back overlay offset down. The offset down from the point of origin where the overlay is printed. The possible values are:

0 – 57.79
: The offset in inches or centimeters, according to the unit of measure value (see page 26-52). The maximum offset is 57.79 centimeters or 22.75 inches.

Bar code. Whether the spooled file contains bar codes created using the BARCODE keyword in the data description specifications (DDS).

For DEVTYPE(*IPDS and *AFPDS), a series of bar code commands is contained in the data stream.

Valid values are Y (yes) or N (no).

Bytes available. The amount of data available to the calling program. In other words, the receiver variable could get this much data from this format if the receiver variable were this large or larger.

Bytes returned. The amount of data returned to the calling program in the receiver variable.

Carriage control characters. Whether the file has machine carriage control characters.

Valid values are Y (yes) or N (no).

Channel value array. Contains the skip-to line number associated with the channel value. (The first entry in the array applies to channel 1.) The variable returns a maximum of 12 binary(4) values. Each of the values returned is 0 if the mode is *NORMAL.

Channel mode. A variable indicating the channel value mode. The possible values are:

**NORMAL*
: The normal channel values (1 equals skip to line 1, 2 through B equal space one line before printing, C equals Skip to overflow line).

blank
: The channel values specified in the channel code array are used.

Character code. Whether the coding for the diskette file is in ASCII or EBCDIC form.

Character ID. Whether the character ID can change within the file.

- For DEVTYPE(*SCS), a Set CGCS through Local ID is contained in the spooled file.
- For DEVTYPE(*IPDS), a Load Font Equivalence command is inserted into the Load Font Equivalence table.
- For DEVTYPE(*AFPDS), a map code font-2 (MCF-2) structured field is contained in the spooled file.

Valid values are Y (yes) or N (no).

Character not valid. Whether incorrect character errors are reported.

Valid values are Y (yes) or N (no).

Character set library name. The name of the library containing the font character set object. The possible values are:

**LIBL*
: The library list is used to locate the font character set object.

library name
: This library is searched for the font character set object.

Character set name. The name of the font character set object used to print this file. The possible values are:

**FONT*
: The information specified on the font parameter is used instead of the character set and code page.

character set name
: The name of the font character set object to use.

Characters per inch. The number (in tenths) of characters per horizontal inch, defined in the printer file. The value 100 indicates 10 characters per inch.

For DEVTYPE(*SCS), a Set Character Distance command is contained in the spooled file.

Characters per inch changes. Whether the spooled file changes the characters per inch (cpi) within the file.

For DEVTYPE(*SCS), a Set Character Distance command is contained in the spooled file.

Valid values are Y (yes) or N (no).

Code page. The mapping of graphic characters to code points for this printer. For *DEVD, the system gets the code page from the printer device description.

For DEVTYPE(*SCS), a Set CGCS through GCID command is contained in the spooled file.

Code page library name. The name of the library containing the code page used to print this spooled file. The possible values are:

**LIBL*
: The library list is used to locate the code page.

library name
: This library is searched for the code page name.

Code page name. The name of the code page used to print this spooled file.

Code pages are groups of characters. Within a code page unique hexadecimal identifiers are assigned to each of the characters.

Coded font array. The name of the fonts used for printing a line data file. This corresponds to the CHARS parameter of MVS and VM printing services. The variable returns a maximum of 4 four-character-coded fonts. The variable is returned blank if no coded fonts are specified. The name specified does not include the two-character prefix of the coded font name (X0 through XG). For more information on the CHARS fields, see the *Print Services Facility/MVS Application Programming Guide* or *Print Services Facility User's Programmers Guide for VM*.

Coded font library name. The name of the library containing the coded font used to print this spooled file. The possible values are:

**LIBL*	The library list is used to locate the coded font.
library name	This library is searched for the coded font.

Coded font name. The name of the coded font used to print this spooled file. A **coded font** is an AFP resource composed of a character set and a code page. The possible values are:

**FNTCHRSET*
: The values used are the values specified on the character set name and library name and code page name and library name fields.

coded font name
: The name of the coded font used to print this spooled file.

Color. Whether an IPDS Write Text command or AFPDS presentation text data (PTX) structured field containing a Set Text Color text control is contained in the spooled file.

For DEVTYPE(*IPDS and *AFPDS), a Set Text Color text control is contained in the spooled file.

Valid values are Y (yes) or N (no).

Control character. Whether this printer file uses the American National Standard printer control character. The possible values are:

**NONE*	No print control characters are passed in the data that is printed.
**FCFC*	The first character of every record is an American National Standard printer control character.

Copies left to produce. The remaining number of copies to be produced on the printer. Valid values are 1 through 255.

CPA3353 message. Whether message CPA3353 is issued when this file is printed.

Valid values are Y (yes) or N (no).

Date file opened. The date that the file was opened in the CYYMMDD format where:

C	Century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
YY	Year
MM	Month
DD	Day

DBCS character rotation. Whether this printer file causes the double-byte character set (DBCS) characters to be rotated 90 degrees counterclockwise before printing. The possible values are *YES and *NO.

DBCS character rotation commands. Whether the double byte character set characters are rotated 90 degrees counterclockwise before printing.

- For DEVTYPE(*SCS), a Set Text Orientation command is contained in the spooled file.
- For DEVTYPE(*AFPDS), a map coded font-1 (MCF-1) structured field, that has the character rotation parameter set to hex 8700 in the spooled file.

Valid values are Y (yes) or N (no).

DBCS characters per inch. The number of double-byte characters to be printed per inch. The possible values are:

-1	*CPI: One-half of the characters per inch value.
-2	*CONDENSED: 20 double-byte characters per 3 inches.
5	5 characters per inch.
6	6 characters per inch.
10	10 characters per inch.

DBCS-coded font library name. The name of the library containing the DBCS-coded font. The possible values are:

**CURLIB*	The current library is searched for the DBCS-coded font.
**LIBL*	The library list is used to locate the DBCS-coded font.
library name	This library is searched for the DBCS-coded font.

DBCS-coded font name. The name of the DBCS-coded font used to print DBCS-coded data on printers configured as AFP(*YES). The possible values are:

**SYSVAL*
: The DBCS-coded font specified in the system value is used.

DBCS-coded font name
: The name of the DBCS-coded font being used.

DBCS data. Whether the file can contain double-byte character set (DBCS) data. The options are *YES or *NO.

DBCS extension characters. Whether the system uses the extension character processing function. The possible values are:

**YES*	The system processes DBCS extension characters.
**NO*	The system does not process DBCS extension characters; it prints extension characters as the undefined character.

DBCS shift-out shift-in (SO/SI) spacing. The presentation of shift-out and shift-in characters when printed. The possible values are:

**YES*	Shift-out and shift-in characters occupy one space.
**NO*	Shift-out and shift-in characters occupy no space.
**RIGHT*	Shift-out characters occupy no space; shift-in characters occupy 2 spaces.

DDS. Whether the file was constructed with DDS.

Valid values are Y (yes) or N (no).

Define characters. Whether the spooled file defines or redefines unique print characters.

For DEVTYPE(*SCS), a Load Alternate Characters command is contained in the spooled file.

Valid values are Y (yes) or N (no).

Device file library name. The name of the library that contains the device file.

Device file name. The name of the device file used to create the spooled file.

Device type. The type of device file for which this file is intended. The possible values are PRINTER, DISKETTE, or TAPE.

Document name. The name of the document that was the source of the spooled file.

Double-wide characters. Whether the spooled file prints everything twice as wide as normal.

For DEVTYPE(*SCS), a Set Font Size command is contained in the data stream.

Valid values are Y (yes) or N (no).

Drawer change. Whether the printer drawer is changed within the spooled file.

- For DEVTYPE(*SCS), a Page Presentation Media command is contained in the spooled file.
- For DEVTYPE(*IPDS), an Execute Order Home State - Select Input Media Source is in the spooled file.
- For DEVTYPE(*AFPDS), a media map and an invoke media map are in the spooled file.

Valid values are Y (yes) or N (no).

Ending page. The page at which printing is to end for the file. 0 or 2147483647 indicates the last page. To print the entire file, set this value to 0, 2147483647, or the last page number.

Exchange type. The exchange type of the diskette file. The possible values are:

**STD*	The system allows the exchange type based on the diskette type and sector size.
**BASIC*	The basic exchange type.
**H*	The H exchange type.
**I*	The I exchange type.

Extended code page. Whether the extended code page changes within the file.

- For DEVTYPE(*SCS), a Set CGCS through GCID is contained in the spooled file.
- For DEVTYPE(*IPDS), a Load Font Equivalence command is inserted into the Load Font Equivalence table.
- For DEVTYPE(*AFPDS), a map code font-2 (MCF-2) structured field is contained in the spooled file.

Valid values are Y (yes) or N (no).

FFT emphasis. Whether the spooled file contains text that is to appear darker than the surrounding text.

For DEVTYPE(*SCS), Begin Emphasis and End Emphasis commands are contained in the spooled file.

Valid values are Y (yes) or N (no).

Field outlining. Whether the spooled file outlines fields of data with boxes. If it does, the file must be printed on a printer that supports field outlining.

For DEVTYPE(*SCS), a Define Grid Line command is contained in the spooled file.

Valid values are Y (yes) or N (no).

File available. The time when this file becomes available to an output device for processing. The possible values are:

**IMMED*	The file is available as soon as the file is opened.
**FILEEND*	The file is available as soon as the file is closed.
**JOBEND*	The file is available when the job that created the file is completed.

File label identifier. The diskette label used when the system last saved the object.

File open. Whether the file is still open when fields are retrieved.

Valid values are Y (yes) or N (no).

File stopped on page boundary. Whether the file has stopped printing on a page boundary.

Valid values are Y (yes) or N (no).

Final form feed. Whether the Final Form Feed command is in the printer file.

Valid values are Y (yes) or N (no).

Final form text. Whether this spooled file contains various functions that are supported on letter-quality printers.

- For DEVTYPE(*SCS), a Document Content Architecture (DCA) command, is contained in the spooled file. DCA commands include:
  - Required New line
  - Required Form Feed
  - Indent Tab
  - Set Presentation Page Size
  - Set Horizontal Margins

- Set Vertical Margins
- Release Left Margin
- Set Line Spacing
- Set Single Line Distance
- Justify Text Field Format
- Set Justify Mode
- Set Horizontal Tab Stops
- Set Indent Level
- Set Exception Action
- Set Presentation Color
- Set Spacing Variable
- Begin and End Overstrike
- Begin and End Underscore
- Bell
- Switch
- Repeat
- Tab
- Backspace
- Unit Backspace
- Substitute
- Word Underscore

Valid values are Y (yes) or N (no).

Fold records. Whether records exceeding the printer forms width are folded (wrapped) to the next line. The possible values are *YES or *NO.

Folder name. The name of the folder that contains the source document.

Font. Whether the spooled file uses multiple fonts.

- For DEVTYPE(*SCS), a Set Font Global command is in the spooled file.
- For DEVTYPE(*IPDS), a Load Font Equivalence command is contained in the Load Font Equivalence table.
- For DEVTYPE(*AFPDS), a map coded font-1 (MCF-1) or map coded font-2 (MCF-2) structured field is in the spooled file.

Valid values are Y (yes) or N (no).

Font equivalence array. The data portion of the Load Font Equivalence (LFE) command for intelligent printer data streams (IPDSs). The variable returns a maximum of 72 sixteen-character font equivalence entries. The current maximum is 48. The additional array space is provided for further expansion. If more than 72 font equivalence entries are used, a 1 is returned in the 1153rd character of the variable; otherwise, the 1153rd character is blank. The format of this entry is described in the *Intelligent Printer Data Stream Reference*.

Form definition library name. The name of the library that contains the form definition.

Form definition name. The name of the form definition to use for this print request.

Form feed. The manner in which forms feed to the printer. The possible values are:

*CONT	Continuous forms
*CUT	Manually fed cut forms
*AUTOCUT	Automatically fed cut forms
*DEVD	As defined in the device description

Form type. The type of form to be loaded in the printer to print this file.

Format name. The name of the format used to return information.

Front margin offset across. For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are:

0 – 57.79	The offset in inches or centimeters, according to the unit of measure value (see page 26-52). The maximum offset is 57.79 centimeters or 22.75 inches.

Front margin offset down. For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are:

-2	*DEVD. For printers configured AFP(*YES), no print border is used for front margin offsets across and down. Otherwise, the offset values are 0.
0 – 57.79	The offset in inches or centimeters, according to the unit of measure value (see page 26-52). The maximum offset is 57.79 centimeters or 22.75 inches.

Front overlay library name. The name of the library containing the front overlay. The possible values are:

*CURLIB	The current library for the job locates the front overlay.
*LIBL	The library list locates the front overlay.
library name	This library is searched for the front overlay.

Front overlay name. The name of the front overlay (the material that prints on the front side of each page). The possible values are:

*NONE	The file does not use the front overlay.
front overlay name	The name of the front overlay.

Front overlay offset across. The offset across from the point of origin where the overlay is printed. The possible values are:

0 – 57.79	The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Front overlay offset down. The offset down from the point of origin where the overlay is printed. The possible values are:

0 – 57.79 The offset in inches or centimeters, according to the unit of measure value (see page 26-40). The maximum offset is 57.79 centimeters or 22.75 inches.

Graphic character set. The set of graphic characters to be used when printing this file. For *DEVD, the system gets the graphic character set from the printer device description.

Graphics. Whether the spooled file contains graphics.

Valid values are Y (yes) or N (no).

Graphics error actions. Whether the file contains graphic error action commands.

For SCS files, the file contains one or more Set Graphic Error Action commands.

Valid values are Y (yes) or N (no).

Graphics token. The printer type on which the graphics in this file can be printed. The possible values are:

4214 Data stream contains graphics for a 4214 printer.
4234 Data stream contains graphics for a 4234 SCS printer.
522X Data stream contains graphics for a 522X printer.
IPDS Data stream contains graphics for an IPDS printer.

Height of drawer 1. The height in inches of the paper in drawer 1. This field is for internal use and should be set to 0 by an application building this format as opposed to retrieving the fields.

Height of drawer 2. The height in inches of the paper in drawer 2. This field is for internal use and should be set to 0 by an application building this format as opposed to retrieving the fields.

Highlight. Whether the spooled file contains text that is to appear darker than the surrounding text.

- For DEVTYPE(*SCS), data is overprinted to get the bold effect.
- For DEVTYPE(*IPDS), a Load Font Equivalence command is contained in the Load Font Equivalence table, which has the bold field set on.
- For DEVTYPE(*AFPDS), a map coded font-2 (MCF-2) structured field that has the font weight class parameter set to hex 07 is in the spooled file.

Valid values are Y (yes) or N (no).

Hold file before written. Whether the file is held. The hold parameter handles this function on a Create Printer File (CRTPRTF), Override Printer File (OVRPRTF), or Change Printer File (CHGPRTF) command. The possible values are:

**YES* The file is held.
**NO* The file is not held.

Internal job identifier. The internal identifier for the job. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Internal spooled file identifier. The value used as input to other programs to improve the performance of locating the spooled file on the system. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

IPDS transparent data. Whether the file contains data from System/36 PRPQs.

Valid values are Y (yes) or N (no).

Job name. The name of the job that created the spooled file.

Job number. The number of the job that created the spooled file.

Justification. The percentage that the output is right-justified. The possible values are 100, 50, or 0.

Last page printed. The number of the last printed page in the file if printing ended before the job completed processing.

Length of page. The length of a page. Units of measurement are specified in the measurement method field.

Lengths present. Whether the 8-byte length information is present in the print text data buffers for format SPLF0200.

Valid values are Y (yes) or N (no).

Line spacing. How a file's line data records are spaced when printed. This information is returned only for *LINE and *AFPDSLINE printer device type files. The possible values are:

**SINGLE* Single-spaced
**DOUBLE* Double-spaced
**TRIPLE* Triple-spaced
**CTLCHAR* The control character field determines line spacing. The control character field can have one of these values:
  **NONE* No print control characters are passed in the data that is printed.
  **FCFC* The first character of every record is an American National Standards printer control character.

Lines per inch. The number (in tenths) of lines per vertical inch defined in the printer file. A value of 40 indicates 4 lines per inch.

Lines per inch changes. Whether the lines per inch (lpi) changes within the spooled file.

- For DEVTYPE(*SCS), a Set Single Line Density command is contained in the spooled file.

- For DEVTYPE(*IPDS), a Load Page Descriptor is contained in the spooled file, which specifies the line-per-inch value
- For DEVTYPE(*AFPDS), a presentation text description (PTD) structured field that specifies how the baseline LPI value changes, is contained in the spooled file.

Valid values are Y (yes) or N (no).

Lines-per-inch (lpi) value not supported. The lines-per-inch (lpi) value is represented in a 1440th-inch value and is not equivalent to 4, 6, 8, 9, or 12. This field applies only to OfficeVision/400 files. It is N for all other spooled files.

Valid values are Y (yes) or N (no).

Maximum forms width. The maximum forms width (in positions) as specified on the printer file.

Maximum records. The maximum number of records allowed in the file at the time the file was opened. A value of 0 indicates no maximum.

Maximum spooled data record size. The length of the largest record in the file. For LINE, AFPDSLINE, and AFDPS files, this length is the length of the largest record in the file. For all other file types, this length is the same as the spooled file buffer size.

Measurement method. The measurement method used for the length of page and width of page fields. The possible values are:

**ROWCOL* Uses rows and columns as the units of measure.

**UOM* Uses the value specified on the unit of measurement parameter (UOM). UOM is either inches (*INCH) or centimeters (*CM).

Multiple up (pages per side). The number of logical pages that print on each side of each physical page when this file is printed. The possible values are 1, 2, and 4.

Number of buffers. The number of buffers currently in the file. If the file is still open when the fields are retrieved, the number of buffers is the number of buffers written to the file thus far.

Number of font array entries. The number of font equivalence array entries used. (The maximum number of font equivalence entries is currently 48.)

Number of resource library entries. The number of entries in the resource library array used. (The maximum number of resource library entries is currently 43.)

Number of separators. The number of file separator pages placed at the beginning of each copy of this file.

OfficeVision/400. Whether the spooled file is generated by the OfficeVision/400 licensed program.

Valid values are Y (yes) or N (no).

Output priority. The priority of the output file. The priority ranges from 1 (highest) to 9 (lowest).

Output queue library name. The name of the library that contains the output queue.

Output queue name. The name of the output queue where the file is located.

Overflow line number. The last line to be printed before the data being printed overflows to the next page.

Page count estimated. Whether the total number of pages is actual or an estimated count.

Valid values are Y (yes) or N (no).

Page definition library name. The name of the library in which the page definition resides. This information is returned only for *LINE or *AFPDSLINE printer device type files.

Page definition name. The name of the page definition to use for the file. This information is returned only for *LINE or *AFPDSLINE printer device type files.

Page length. The page length (in lines per page) used by the spooled file. The valid range is row 1 through 255. The value used should not exceed the actual length of the page used.

Page or record being written. The page number or record number currently being written. The page number may be lower or higher than the page number actually being printed because of buffering done by the system. The page number shown may be zero if:

- The printer file is routed to a diskette unit.
- The writer is currently printing job or file separators for the file.

Page position. Whether page positioning errors are reported.

Valid values are Y (yes) or N (no).

Page rotate. Whether the spooled file changes the page rotation to be used within the file.

- For DEVTYPE(*SCS), a Set Text Orientation command is contained in the spooled file.
- For DEVTYPE(*IPDS), a Load Page Descriptor command is contained in the spooled file. See the *Intelligent Printer Data Stream Reference* for detailed information about rotating text in a spooled file.
- For DEVTYPE(*AFPDS), a presentation text descriptor (PTD) structured field that specifies the page rotation is contained in the spooled file.

Valid values are Y (yes) or N (no).

Page rotation. The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. The possible values are:

-1 *AUTO: Computer output reduction is done automatically if the output is too large to fit on the form, regardless of the print quality.

-2 *DEVD: Page rotation is obtained from the printer device description.

-3	*COR: Output created for a form 13.2 inches wide by 11.0 inches long is adjusted to print on a form 11.0 inches wide by 8.5 inches long.
0	No rotation is done. Printing starts at the edge loaded into the printer first, and is parallel to that edge.
90	Text is rotated 90 degrees clockwise from the 0-degree writing position.
180	Text is rotated 180 degrees clockwise from the 0-degree writing position.
270	Text is rotated 270 degrees clockwise from the 0-degree writing position.

Page width. The page width (in characters per printed line) used by the spooled file. The valid range is column 1 through 378, although some printers have a page width less than 378. The value should not exceed the actual width of the page used.

Point size. The point size in which this file's characters should be printed.

Print fidelity. The kind of error handling that is performed when printing. The possible values are:

**ABSOLUTE*	The file is printed only if it can be printed exactly as specified in the data stream.
**CONTENT*	The printing overrides errors in the data stream and continues printing with the printers best quality based on the content fidelity.

Print on both sides (duplex). How the information prints. The possible values are:

**FORMDF*	The file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, and *AFPDSLINE printer device type files.
**NO*	The printing on the page is on one side only.
**YES*	The printing is on both sides of the page with the top of each page the same for both sides.
**TUMBLE*	The printing is on both sides with the top of one printed page at the opposite end from the top of the other printed page.

Print quality. The print quality that is used when printing this output. The possible values are:

**STD*	Standard
**DRAFT*	Draft
**NLQ*	Near-letter quality
**FASTDRAFT*	Prints at a faster speed than *DRAFT

Print text. The text that is printed at the bottom of each page of printed output and on separator pages.

Printer device type. The type of data stream used to represent the file. The possible values are:

**AFPDS*	Advanced Function Printing data stream
**AFPDSLINE*	AFPDS data mixed with 1403 line data
**IPDS*	Intelligent printer data stream
**LINE*	1403 line data
**SCS*	Systems Network Architecture (SNA) character stream
**USERASCII*	ASCII data

Printer font. The printer font used. If *DEVD is shown, the printer uses the font defined in the printer device description. If *CPI is shown, the file is printed with a font that has the pitch specified by the CPI (character per inch) field.

Program that opened file library name. The name of the library that contains the program that opened the file.

Program that opened file name. The name of the program that opened the spooled file.

Record format. The format of the records. The possible values are:

**FIXED*	All records in the file are the same size; that is, the original length field of the print data is the same for all records in the file.
**VARIABLE*	Not all records in the file are the same size; that is, the original length field of the print data is the not the same for all records in the file.

Record length. The length of the file's records. If the field shows -1 (the special value for *RCDFMT), this is an externally defined file, and the length is included in the file definition. The length includes the extra length of 1 for carriage controls.

Replace unprintable characters. Whether characters that cannot be printed are to be replaced with another character. The options are Y (yes) or N (no).

Replacement character. The character that replaces any unprintable characters. This field has a value if the replace unprintables field specifies Y.

Reserved. Reserved for byte alignment.

Resource library array. The library names in the library list to use when the spooled file is printed. The variable returns a maximum of 63 ten-character library names. The current maximum is 43. The additional array space is provided for further expansion. If more than 63 resource libraries are used, a 1 is returned in the 631st character of the variable; otherwise, the 631st character is blank.

Restart printing. The number of the page where printing restarts. When you specify a value while the file is printing, the writer stops printing and restarts on the specified page. If the file is not currently printing, this change takes effect after the first copy prints. The possible values are:

-1	*STRPAGE: The starting page specified in the PAGERANGE parameter is the page on which to restart printing.
-2	*ENDPAGE: Only the last page prints.
-3	*NEXT: The next page of the file to print is the page where printing is restarted.

| *restart page* The page number you specify is where printing restarts.

| **Save file after written**. Whether this file is to be saved after it is written. The possible values are *YES and *NO.

| **SCS data**. Whether the spooled file is created with SCS already in the input data.

| Valid values are Y (yes) or N (no).

| **Set exception**. Whether the file has the SCS Set Exception Action (SCS) command in the data stream.

| Valid values are Y (yes) or N (no).

| **Set Line Density command**. Whether the lines per inch (lpi) changes within the spooled file or is specified with the Set Line Density SCS command.

| For DEVTYPE(*SCS), a Set Line Density command is contained in the spooled file.

| Valid values are Y (yes) or N (no).

| **Source drawer**. The drawer to be used when the automatically cut form feed option is selected. The possible values are:

| *1-255* The drawer number.
| *-1* *E1, the envelope drawer.
| *-2* *FORMDF, indicating that the file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, or *AFPDSLINE printer device type files.

| **Spooled file buffer size**. The maximum size of a spooled file buffer. Valid lengths are 512 and 4079.

| **Spooled file level**. The level of the spooled file in Version, Release, and Modification level format (VxRxMx).

| **Spooled file name**. The name of the spooled file whose information is retrieved.

| **Spooled file number**. The spooled file number of the specified file.

| **Starting page**. The page at which printing is to start for the file. The possible values are:

| *0* Printing starts on page 1.
| *-1* Use the ending page value.
| *1* The entire file prints.

| **Status**. The status of the file is one of the following values:

| **CLOSED* The program completely processed the file, but SCHEDULE(*JOBEND) was specified, and the job that produced the file has not yet finished.
| **HELD* The file is held.
| **MESSAGE* The file has a message requiring a reply or an action.
| **OPEN* The file was not completely processed and is not ready for a writer.
| **PENDING* The file is pending to be printed.
| **PRINTER* The complete file was sent to the printer. but a print complete status was not sent back.
| **READY* The file is available to be written.
| **SAVED* The file was written and is saved. This file remains saved until it is released.
| **WRITING* The writer is writing the file on a diskette or a printer device.

| **Subscript**. Whether the spooled file contains subscript.

- For DEVTYPE(*SCS), a Subscript command is contained in the spooled file.
- For DEVTYPE(*IPDS and *AFPDS), a Temporary Baseline Move text control is contained in the spooled file.

| Valid values are Y (yes) or N (no).

| **Superscript**. Whether the spooled file contains superscript.

- For DEVTYPE(*SCS), a Superscript command is contained in the spooled file.
- For DEVTYPE(*IPDS and *AFPDS), a Temporary Baseline Move text control is contained in the spooled file.

| Valid values are Y (yes) or N (no).

| **System/36 procedure name**. The name of the procedure running when the spooled file was created.

| **System/36 spooled file identifier**. The 6-character identifier assigned to the spooled file.

| **System/38 text utility flags**. The flag for advanced functions.

| When you create a file that will contain data that has never been spooled before, set the field to hexadecimal zeros (hex 00).

| **Time file opened**. The time that the file was opened in the HHMMSS format where:

| *HH* Hour
| *MM* Minutes
| *SS* Seconds

| **Total copies**. The total number of copies to be produced for this printer file.

| **Total pages**. The number of pages this printer file contains.

| **Total records**. If the spooled file data stream is *AFPDS, *AFPDSLINE, or *LINE or the spooled file device type is diskette, this field contains the total number of records in the printer file or the diskette file. The field is blank if the file is open.

| **Transparency**. Whether the SCS Transparency command is used in the spooled file. It is set to N for all other device type files, such as IPDS.

| For DEVTYPE(*SCS), a TRANSPARENT SCS command is contained in the spooled file.

Valid values are Y (yes) or N (no).

TRC for 1403. Whether the file contains 1403 line data with table-reference characters (TRC). This field is only used for device types *LINE and *AFPDSLINE.

Valid values are Y (yes) or N (no).

Unit of measure. The unit of measure to use for specifying distances. The unit of measure is used with certain parameters of the printer file. Page definitions and form definitions are examples of these parameters. The possible values are:

*CM Centimeters
*INCH Inches

Unrecognizable data. Whether the file contains SCS commands that are not valid because of one of the following:

- Command is not recognized
- Command data is not valid
- Command is recognized, but not supported

Valid values are Y (yes) or N (no).

User name. The name of the user that created the spooled file.

User-defined file. Whether the spooled file was created by using an API. The possible values are:

*YES The spooled file was created using the QSPCRTSP API.
*NO The spooled file was created by normal system functions.

User-generated data stream. Whether the data stream has been validated by a system program on the AS/400 system when the file was spooled, for example, the OfficeVision/400 program.

Valid values are Y (yes) or N (no).

User-specified data. The 10 characters of user-specified data that describe the file.

Volumes (array). The diskette volumes used for saving the object. The variable returns a maximum of 10 six-character volumes. The volume IDs begin in character positions 1, 8, 15, 22, 29, 36, 43, 50, 57, and 64. If more than 10 volumes are used, a 1 is returned in the 71st character of the variable; otherwise, the 71st character is blank. The variable is returned blank if the object was saved to a save file, or if it was never saved.

Width of drawer 1. The width in inches of the paper in drawer 1. This field is for internal use and should be set to 0 by an application building this format as opposed to retrieving the fields.

Width of drawer 2. The width in inches of the paper in drawer 2. This field is for internal use and should be set to 0 by an application building this format as opposed to retrieving the fields.

Width of page. The width of a page. Units of measurement are specified in the measurement method field.

3812 SCS. Whether the spooled file contains fonts supported only on the 3812 printer.

For DEVTYPE(*SCS), a Set Coded Font Local command with a global ID of greater than 255 is contained in the spooled file.

Valid values are Y (yes) or N (no).

3812 SCS commands. Whether the file contains commands that are not valid on the 3812 printer.

- An incorrect Set Character Set Global ID is found by validity checking the code page with what is valid for the 3812 printer.
- An incorrect Set Character Set Local ID is found by validity checking the code page with what is valid for the 3812 printer.
- A Set Presentation Color command

Valid values are Y (yes) or N (no).

5A present. Whether the 5A hexadecimal constant is present in all AFPDS data for the file (in format SPLF0200).

Valid values are Y (yes) or N (no).

5219 commands. Whether the file contains commands that are not valid on a 5219 printer. The data stream contains one or more of the following SCS commands which, either are not valid on a 5219 or contain parameters that are not valid.

- An incorrect Set Character Set Global ID is found by validity checking the code page with what is valid for the 5219 printer.
- Begin and End Emphasis SCS commands.
- A Set Presentation Page Size command contains a page size value that is not supported.
- A Page Presentation Media command contains a drawer value other than drawer 1 or drawer 2.
- A Set Single Line Distance command contains a distance value that is not supported.
- A Set Single Line Spacing command contains a distance value that is not supported.
- A Justify Text Field Format command contains a value other than 0, 50, or 100 percent.
- A Set Justify Mode Format command contains a value other than 0, 50, or 100 percent.
- A Set Presentation Color command.
- A Begin Overstrike command, which contains an optional CHRID value of other than 0.

Valid values are Y (yes) or N (no).

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C20 E Error found by program &1.

CPD3C20 D Error occurred with receiver variable specified.
CPD3C21 D Format name &1 is not valid.
CPD3C24 D Length of the receiver variable is not valid.

| CPD3C40 D Spooled file number &1 is not valid.
| CPD3C42 D User name or job number is not blank.
| CPD3C43 D Internal job identifier is not valid.
| CPD3C44 D Internal spooled file identifier is not valid.
| CPD3C58 D Job name specified is not valid.
| CPD33C9 D Spooled file name parameter cannot be blank.
| CPF3C21 E Format name &1 is not valid.
| CPF3C24 E Length of the receiver variable is not valid.
| CPF3C33 E Spooled file number &1 is not valid.
| CPF3C36 E Number of parameters, &1, entered for this API was not valid.
| CPF3C40 E Spooled file &4 not found.
| CPF3C41 E More than one spooled file with same name.
| CPF3C42 E User name or job number is not blank.
| CPF3C43 E Internal job identifier is not valid.
| CPF3C44 E Internal spooled file identifier is not valid.
| CPF3C58 E Job name specified is not valid.
| CPF33C9 E Spooled file name parameter cannot be blank.
| CPF3309 E No files named &1 are active.
| CPF3330 E Necessary resource not available.
| CPF3342 E Job &5/&4/&3 not found.
| CPF3343 E Duplicate job names found.
| CPF3344 E File &1 number &2 no longer in the system.

Part 14. User Interface APIs

Chapter 27. User Interface APIs

This chapter includes user interface APIs that allow the user to manipulate functions of the interface for an application. These APIs are:

- Display Command Line Window (QUSCMDLN)
- Display Help (QUHDSPH)

Chapter 28, "User Interface Manager APIs" on page 28-1 discusses the user interface management (UIM) APIs.

Display Command Line Window (QUSCMDLN) API

The Display Command Line Window (QUSCMDLN) API displays a window containing a command line. A **window** is an area on the display that is treated as a separate display. Windows have visible boundaries and appear to overlay the display from which they are requested.

From the command line displayed, the user can:

- Enter commands.
- Retrieve commands and run programs.
- Prompt for commands.
- Use help for CL commands.
- Request override commands. When the command line window is removed, all overrides are deleted.

The QUSCMDLN API has no parameters.

For consistency with other AS/400 displays, you should use the F9 key to request a command line window.

The command line window produced by the QUSCMDLN API is shown at the bottom of the following display:

```
CSTINV                    CUSTOMER INVOICE MENU

Select one of the following:

     1. Enter or change orders
     2. Print picking slips
     3. Release orders
     4. Print invoices

.....................................................................
:                              Command                              :
:                                                                   :
: ===> ____________________________________________________________ :
: F4=Prompt   F9=Retrieve   F12=Cancel                               :
:                                                                   :
:...................................................................:
```

Display Appearance Problems

There are two rare situations where portions of the underlying display might not be shown correctly while the command line window is shown. The underlying display is not actually changed, and it is shown correctly once the command line window is removed.

The situations are:

1. The underlying display is a text editor or other display that uses special work station printer mode functions. While the command line window is displayed, the special work station printer mode characters are not shown.
2. The underlying display is a bidirectional right-to-left display. While the command line window is displayed, the underlying display is flipped and shown from left-to-right.

Error Messages

CPF9871 E Error occurred while processing.

Display Help (QUHDSPH) API

Parameters

Required Parameter Group:

1	Help identifier array	Input	Char(*)
2	Number of help identifiers	Input	Binary(4)
3	Help type	Input	Array(2) of Binary(4)
4	Full display title	Input	Char(55)
5	Index search object and library name	Input	Char(20)
6	Display type	Input	Char(1)
7	Upper left corner	Input	Array(2) of Binary(4)
8	Lower right corner	Input	Array(2) of Binary(4)
9	Cursor location	Input	Array(2) of Binary(4)
10	Error code	I/O	Char(*)

The Display Help (QUHDSPH) API displays help information. The help can be either contextual or extended. It can be formatted as a full display or in a window.

Contextual help provides information about a single item, such as the field on which the user's cursor is positioned when help is requested. **Extended help** provides help for all the items on the display; it contains all contextual help items and can contain additional information as well. A **window** is an area on the display that is treated as a separate display. Windows have visible boundaries and appear to overlay the display from which they are requested.

You can use the QUHDSPH API to handle Help key processing in applications written in data description specifications (DDS) or user-defined data streams (UDDS). You do not need it to handle help requests in applications that use record-level DDS display files with DDS help keywords to present panels because the DDS help keywords handle all help requests.

You do not need to use this API to display help for applications written using panel groups to present panels, because the UIM handles all help requests.

Authorities and Locks

Library Authority	*READ
Index Search Object Authority	*USE
Index Search Object Library Authority	*READ
Index Search Object Lock	*SHRRD
Panel Group Authority	*USE
Panel Group Lock	*SHRRD

Required Parameter Group

Help identifier array
INPUT; CHAR(*)
An array of the help identifiers to display. The list can contain up to 2000 items. Each item has two parts:

Help panel group and library name
CHAR(20)
The panel group (*PNLGRP) object that contains the help module to display, and the library in which it is located. (A **panel group** is an object with an object type of *PNLGRP. It contains display panels, print panels, or help modules.)

The first 10 characters contain the panel group object name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Help module name
CHAR(32)
The name specified on the NAME attribute of a :HELP. tag in the panel group object. The name must be specified using uppercase, alphabetic characters.

Number of help identifiers
INPUT; BINARY(4)
The number of help identifiers in the help identifier array parameter. The number must be between 1 and 2000.

Help type
INPUT; ARRAY(2) of BINARY(4)
Whether this is a request to display extended or contextual help, and which help identifiers to display for the latter. For contextual help, only a subset of the help identifiers in the help identifier array parameter is initially displayed, and a function key is enabled to display extended help. Extended help includes all help identifiers in the help identifier array parameter, including those used in contextual help.

This parameter is a 2-element array of BINARY(4) values. Both elements are used as indexes into the help identifier array. The array elements are:

1. The first help identifier to display for contextual help. The value must be greater than or equal to 1, and less than or equal to the number of help identifiers in the help identifier array.
2. The last help identifier to display for contextual help. The value must be greater than or equal to the value specified in the first element of this parameter, and less than or equal to the number of help identifiers in the help identifier array.

To display extended help, set the value of the first array element to 1 and the value of the second array element to the value specified in the number of help identifiers parameter.

Full display title
INPUT; CHAR(55)
The default title to use if the help is shown in a full display and if no title is found in the help panel group object.

Index search object and library name
INPUT; CHAR(20)
The index search (*SCHIDX) object that can be accessed from the help display, and the library in which it is located. The first 10 characters contain the index search object name, and the second 10 characters contain the library name.

You can use the following special value for this parameter:

***NONE** The index search function is not made available for this help request.

You can use the following special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Display type
INPUT; CHAR(1)
Whether the help information is displayed in a full screen or a window. You must use one of the following values:

Y The help is displayed in a full screen.
N The help can be displayed in a window, depending on the user option (USROPT) value of the user profile.

Upper left corner
INPUT; ARRAY(2) of BINARY(4)
The upper left corner of the area on the display for which help is requested. If the help is displayed in a window, the window is adjacent to the area identified, if possible. The two array elements are:

1. The row number of the upper left corner
2. The column number of the upper left corner

Lower right corner
INPUT; ARRAY(2) of BINARY(4)
The lower right corner of the area on the display for which help is requested. If the help is displayed in a window, the window is adjacent to the area identified, if possible. The two array elements are:

1. The row number of the lower right corner
2. The column number of the lower right corner

Cursor location
INPUT; ARRAY(2) of BINARY(4)
The position of the cursor when help is requested. If the help is displayed in a window, this cursor position is used by the UIM to decide the position and size of the window. The array elements are the row number of the cursor position and the column number of the cursor position.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6E00 E All CPF6EXX messages could be signalled. XX is from 01 to FF.

Chapter 28. User Interface Manager APIs

The user interface manager (UIM) APIs allow an application developer to manipulate the user interface. These APIs are used in combination with variables, lists, and panel definitions in a panel group object. For more information on creating panel group objects, see the *Guide to Programming Displays*.

The APIs and their parameter descriptions are listed in alphabetical order throughout this chapter. The following list is a task-oriented summary of the categories provided by the UIM:

- Application APIs:

  Close Application (QUICLOA)
  Open Display Application (QUIOPNDA)
  Open Print Application (QUIOPNPA)

- Display APIs:

  Add Pop-Up Window (QUIADDPW)
  Display Panel (QUIDSPP)
  Remove Pop-up Window (QUIRMVPW)
  Set Screen Image (QUISETSC)

- List APIs:

  Add List Entry (QUIADDLE)
  Add List Multiple Entries (QUIADDLM)
  Delete List (QUIDLTL)
  Get List Entry (QUIGETLE)
  Get List Multiple Entry (QUIGETLM)
  Remove List Entry (QUIRMVLE)
  Retrieve List Attributes (QUIRTVLA)
  Set List Attributes (QUISETLA)
  Update List Entry (QUIUPDLE)

- Print APIs:

  Add Print Application (QUIADDPA)
  Print Panel (QUIPRTP)
  Remove Print Application (QUIRMVPA)

- Variable pool APIs:

  Get Dialog Variable (QUIGETV)
  Put Dialog Variable (QUIPUTV)

Terms and Definitions

Application variable pool. The set of all dialog variable values for an open application.

Argument list. In UIM, this list consists of values that are passed to a program.

Contextual help. Help information about a single item, such as the field on which the cursor is positioned when help is requested.

Dialog variable. A named element in a panel group used to pass data values between programs or between a program and a user. The current contents of all dialog variables corresponding to variables in the list are saved as each entry is added.

Error variable. The dialog variable specified on the ERRVAR attribute of the variable definition (VAR) tag. The error variable is used to set and test the error status of the dialog variable named on the NAME attribute of the VAR tag.

Extended action entry. The first line below the column headings, containing the action option column and entry-capable fields for one or more additional list columns.

Extended action list area. A list area of a panel that contains an extended action entry.

Extended help. Help information for all the items on the display; it contains all contextual help items and can contain additional information as well.

List entry handle. A value that uniquely distinguishes an entry in a UIM list until it is removed from the list. A list entry handle is meaningful only for a particular open application, list, and entry combination. It has no meaning in any other open UIM application, or even in the same application if the list or the entry is deleted and then re-created. Unpredictable results are possible if a list entry handle is used outside of this definition.

Message reference key. A unique string of characters that identify a particular instance of a message in a queue. The message key can also be used to refer to a specific instance of a message in order to move, receive, reply to, resend, or move it.

Open data path (ODP). A control block containing information about the merged file attributes and information returned by I/O operations.

Pop-up window. An area of the screen with visible borders, which supplements the dialog occurring in the full-screen panel or in a previous pop-up window.

Pull-down field choice. A choice that appears in a pull-down menu.

Trimming. An operation performed by removing a list entry from the end of the list opposite from the end where the new entry is added.

Variable buffer. A buffer used to pass dialog variables between the application program and the UIM.

Variable record. A named element of a panel group that identifies the content and layout of a buffer of dialog variables.

Window. An area on the display that is treated as a separate display. Windows have visible boundaries and appear to overlay the display from which they are requested.

Add List Entry (QUIADDLE) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Input	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Option	Input	Char(4)
7	List entry handle	Output	Char(4)
8	Error code	I/O	Char(*)

The Add List Entry (QUIADDLE) API adds one new entry to a list. The new entry is inserted immediately before or immediately after the entry identified by the current entry pointer for the list. The new entry can also be inserted at the beginning or at the end of the list. On return to the application program, the current entry pointer points to the newly inserted entry.

Required Parameter Group

Application handle
INPUT; CHAR(8)
A value assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
INPUT; CHAR(*)
Program storage containing values of one or more dialog variables, from which dialog variable values are copied. Dialog variables are copied in the order specified in the variable record definition.

If the variable record name parameter specifies the name of a variable record defined in the panel group for an open application, dialog variables are copied from the variable buffer to the application variable pool before the list entry is added. This parameter operation is the same as using the Put Dialog Variable (QUIPUTV) API immediately before the QUIADDLE API.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the variable record definition specified in the variable record name parameter. If the buffer is not large enough, an error condition is reported.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied from the variable buffer to the application variable pool. The variable record must be defined in the panel group for the open application. The following special value can be used:

***NONE** The QUIPUTV API is not used during the QUIADDLE API. The variable buffer parameter and variable buffer length are ignored.

List name
INPUT; CHAR(10)
The name of the list to which an entry is added. If the list is not currently active in the open application, it is activated by this API.

Option
INPUT; CHAR(4)
The location of the new entry in the list. When an entry is added to the list, the current entry pointer is always changed to point to the new list entry.

One of the following values must be specified to indicate the new entry's location:

FRST Added as the first entry in the list
LAST Added as the last entry in the list
NEXT Added after the current entry
PREV Added before the current entry

List entry handle
OUTPUT; CHAR(4)
A value representing an entry in a UIM list and returned to the application program representing the current entry in the list until it is removed from the list, even if other entries are inserted and removed from the list. This value is the handle of the entry just inserted in the list.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6AA1 E The value of the action field is not correct at this time. Reason code &5.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A2B E Value for Option parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A36 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A37 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A9D E Size limit reached for list &4.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A93 E Operation not valid when current entry is &5.

Add List Multiple Entries (QUIADDLM) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Input	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Option	Input	Char(4)
7	List entry handle	Output	Char(4)
8	Number of records	Input	Binary(4)
9	Record numbers	Input	Char(*)
10	Record size	Input	Binary(4)
11	Record count	Output	Binary(4)
12	Error code	I/O	Char(*)

The Add List Multiple Entries (QUIADDLM) API adds one or more new entries to a list. The new entries are inserted immediately before or immediately after the entry identified by the current entry pointer for the list. They can also be inserted at the beginning or end of the list. On return to the application program, the current entry pointer points to the most recently inserted entry.

The contents of all dialog variables corresponding to columns in the list are saved in each added entry. When the operation completes successfully, the corresponding dialog variables contain the values from the last list entry successfully added.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
INPUT; CHAR(*)
The program buffer from which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record definition.

If the variable record name parameter specifies the name of a variable record defined in the panel group for an open application, dialog variables are copied from the variable buffer to the application variable pool before the list entry is added. This parameter operation is the same as using the Put Dialog Variable (QUIPUTV) API immediately before the Add List Multiple Entry (QUIADDLM) API.

The variable buffer must be large enough to contain all the variables specified in the variable record definition.

When the number of records parameter has a value greater than one and the record numbers parameter has a value of zero, the size of the variable buffer must be at least equal to the value on the number of records parameter multiplied by the value on the record size parameter.

When the number of records parameter has a value greater than one and the record numbers parameter has a nonzero value, the size of the variable buffer must be at least equal to the largest value in the array of record numbers multiplied by the value of the record size parameter.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the variable record definition specified in the the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record determining which dialog variables are copied from the variable buffer to the application variable pool. The variable record must be defined in the panel group for the open application.

The following special value can be used:

***NONE** The QUIPUTV API is not used during the QUIADDLM API. The variable buffer, variable buffer length, number of records, record size, and record numbers parameters are ignored.

List name
INPUT; CHAR(10)
The name of the list to which entries are added. If the list is not currently active in the open application, it is activated by this API.

Option
INPUT; CHAR(4)
The location of the new entry in the list. When an entry is added to the list, the current entry pointer is always changed to point to the new list entry.

One of the following values must be specified to indicate the new entry's location:

FRST Added as the first entry in the list
LAST Added as the last entry in the list
NEXT Added after the current entry
PREV Added before the current entry

List entry handle
OUTPUT; CHAR(4)
The list entry handle returned to the application program from the current entry pointer. This value is the handle of the entry just inserted into the list. If more than one list entry is added, this parameter contains the list entry handle of the last entry successfully added.

Number of records
INPUT; BINARY(4)
The total number of entries the application program wants to add to the list. If the variable record name parameter specifies the name of a variable record

defined in the panel group for this open application, this parameter can be used with the record size, record count, and record numbers parameters to add multiple entries to the list. When the variable record name parameter has the value *NONE, the number of records parameter is ignored.

The following special value can be used:

1 Only one entry is added to the list. The record size, record count, and record numbers parameters are ignored.

When the number of records parameter is greater than 1, the variable buffer parameter must contain all the entries to be added to the list. The record size parameter defines the size of each record within the variable buffer, and is used to calculate the offset of each record from the first record. The variable record, identified by the variable record name parameter, defines the format of each record.

Record numbers
INPUT; CHAR (*)
An array of record numbers. Each entry in the array is a BINARY(4) value from 1 to 32 767. The array specifies the order in which entries are added to the list. The first record number defines the first entry added to the list, the second record number defines the second entry, and so forth. Each record number specifies a record from the variable buffer containing the values for an entry to be added to the list. The dimension of the array must be at least as large as the value for the number of records parameter.

The following special value can be used in the first array element:

0 Records from the variable buffer are used in sequential order. The array of record numbers needs to have only one element.

When the record numbers parameter is 0, the size of the variable buffer must be at least equal to the value specified on the number of records parameter multiplied by the value on the record size parameter. When the record numbers parameter has a nonzero value, the size of the variable buffer must be at least equal to the largest value specified in the record numbers array multiplied by the record size value.

The record numbers parameter allows applications to add more than one list entry from data structures already available to the application program. This is useful when the existing order of records in the data structures is not desired for the list entries, or when certain records should not be added to the list.

Record size
INPUT; BINARY(4)
The size of each record within the variable buffer when multiple records are added to the list. The record size is also used to calculate the offset of each record after the first record.

When the number of records parameter is 1, this parameter is ignored.

Record count
OUTPUT; BINARY(4)
The number of list entries actually added. Validation is done on packed or zoned data values in each record. If an error is found, the Add List Multiple Entries (QUIADDLM) API ends at that point with an exception; unusable data causes a partial update of the list. This parameter returns the number of list entries successfully added to the application program, even if an error occurs.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6AA1 E The value of the action field is not correct at this time. Reason code &5.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A06 E Record size or record number too large.
CPF6A2B E Value for Option parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A30 E Value for Record Numbers parameter not valid.
CPF6A36 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A37 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A9D E Size limit reached for list &4.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A93 E Operation not valid when current entry is &5.

Add Pop-Up Window (QUIADDPW) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Pop-up window location	Input	Char(10)
3	Row	Input	Binary(4)
4	Column	Input	Binary(4)
5	Error code	I/O	Char(*)

The Add Pop-Up Window (QUIADDPW) API begins the ability to display panels within a pop-up window. All subsequent panel displays for the open application remain in the pop-up window until the Remove Pop-up Window (QUIRMVPW) API is called, or until the QUIADDPW API is called again to add another pop-up window. The maximum

number of pop-up windows that can be added to an open application is 20.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Pop-up window location
INPUT; CHAR(10)
The name of a field used for field-adjacent positioning. Field-adjacent positioning places a pop-up window near a field on the underlying panel. The field name must correspond to a name specified on the NAME attribute on one of the following tags of the immediately underlying panel or pop-up window:

- Command Line (CMDLINE)
- Data Item Group (DATAGRP)
- Data Item (DATAI)
- List Column (LISTCOL)
- List Column Group (LISTGRP)
- Menu Item (MENUI)
- Option Line (OPTLINE)

One of these special values may be used to position the window:

***OFFSET** The pop-up window is offset from the top left corner of the most recent underlying panel or pop-up window.

***PULLDOWN** The pop-up window is positioned adjacent to the previous position of the pull-down choice. If the most recent action for the most recently displayed panel was not the selection of a pull-down choice, offset positioning is used.

***ROWCOL** The row and column parameters indicate where the upper left corner of the pop-up window appears. Row and column positioning should be used only when displaying a pop-up window over a non-UIM panel, in conjunction with the Set Screen Image (QUISETSC) API.

Row
INPUT; BINARY(4)
The absolute row used when row and column positioning is requested. The value is either a positive or negative integer.

If a negative integer is provided, the last row for the pop-up window is calculated using the absolute value of the integer, relative to the bottom of the display. A pop-up window cannot begin in row one, and a minimum-depth pop-up window must fit at the specified first row.

When a negative integer is provided, the absolute location of the window is determined by the size of the first panel displayed in the pop-up window. If another panel is displayed in the same window with a smaller width or depth, the location is not recalculated based on the negative integer. If another panel is displayed in the same window with a larger width or depth, the window is changed to use offset positioning.

Column
INPUT; BINARY(4)
The absolute column used when row and column positioning is requested. The value is either a positive or a negative integer.

If a negative integer is provided, the last column for the pop-up window is calculated using the absolute value of the integer, relative to the right side of the display. A pop-up window cannot begin in column one, and a minimum-width pop-up window must fit at the specified first column.

When a negative integer is provided, the absolute location of the window is determined by the size of the first panel displayed in the pop-up window. If another panel is displayed in the same window with a smaller width or depth, the location is not recalculated based on the negative integer. If another panel is displayed in the same window with a larger width or depth, the window is changed to use offset positioning.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3E E Application not open for display.
CPF6A81 E Window cannot be added at this time.
CPF6A82 E &4 is not a valid window location.
CPF6A83 E Row or column given for positioning is not valid.
CPF6A85 E Attempted to display more than &4 windows at a time.

Add Print Application (QUIADDPA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Qualified printer file name	Input	Char(20)
3	Alternative file name	Input	Char(10)
4	Share open data path	Input	Char(1)
5	User data	Input	Char(10)
6	Error code	I/O	Char(*)

The Add Print Application (QUIADDPA) API enables print functions in a previously opened display application by opening the printer file for the application. The QUIADDPA API and the Remove Print Application (QUIRMVPA) API are used in pairs to add and remove printing from applications.

Because the QUIADDPA API requires an open application for display, this print function does not work in a batch environment. For printing in batch, use the Open Print Application (QUIOPNPA) API.

Authorities and Locks

Library Authority	*READ
Printer Device File Authority	*USE
Printer Device File Lock	*SHRNUP

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Qualified printer file name
INPUT; CHAR(20)
The name of the printer device file used for print operations. The first 10 characters contain the name of the *FILE object, and the second 10 characters contain the name of the library in which the printer device file resides. These special values can be used for the library name:

***CURLIB** The job's current library
***LIBL** The library list

The user must have *USE authority to the file named by this parameter.

Alternative file name
INPUT; CHAR(10)
An alternative name for the spooled output file. The following special value can be used:

***NONE** There is no alternative name for the spooled output file. The name of the spooled output file is the name of the printer device file.

Share open data path
INPUT; CHAR(1)
Indicates whether or not the open data path (ODP) for the printer file is shared. Sharing the ODP allows multiple UIM applications to print to the same spooled output file. One of the following values must be used:

Y The ODP is shared.
N The ODP is not shared.

User data
INPUT; CHAR(10)
User data associated with the spooled output file. This data becomes an attribute of the spooled output file. The following special value can be used:

***NONE** There is no user data associated with the spooled output file.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A1A E Application already has an open print file.
CPF6A1C E Unable to add print function.
CPF6A1E E Object cannot be used with this device or print file.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A20 E Print code page not identical to display code page.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF9850 E Override of printer file &1 not allowed.

Close Application (QUICLOA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Close option	Input	Char(1)
3	Error code	I/O	Char(*)

The Close Application (QUICLOA) API closes a UIM application that was opened using the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API. The open and close APIs must be used in pairs to open and close each UIM application.

The QUICLOA API releases all UIM resources associated with the open application, destroying the variable pool and deleting any associated active lists. The storage for internal control blocks supporting the application is freed, and all locks acquired for the open application are released.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The open application for which this API is called. The application handle is assigned by the UIM and returned to the application program when the application is opened.

Close option
INPUT; CHAR(1)
Indicates whether to perform a normal or abnormal close operation. One of the following values must be specified:

A An abnormal close operation is performed. This operation is used when the application prematurely ends processing in the most efficient manner possible. When an application with an open printer file is closed with the abnormal option, the printer trailer text specified on the print trailer message (PRTTRAIL) tag is not printed.

M A normal close operation is performed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.

Delete List (QUIDLTL) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	List name	Input	Char(10)
3	Error code	I/O	Char(*)

The Delete List (QUIDLTL) API deletes an active list and provides a way for the application to start over with a new list.

QUIDLTL does not need to be used before the Close Application (QUICLOA) API because all lists associated with an open application are automatically deleted when the application is closed.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or Open Print Application (QUIOPNPA) API when the application is opened.

List name
INPUT; CHAR(10)
The name of the list to be deleted. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) API, or its attributes are set with the Set List Attributes (QUISETLA) API.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.

Display Panel (QUIDSPP) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Function requested	Output	Binary(4)
3	Panel name	Input	Char(10)
4	Redisplay option	Input	Char(1)
5	Error code	I/O	Char(*)

Optional Parameter Group:

6	User task	Input	Char(1)
7	Program stack counter	Input	Binary(4)
8	Program message queue	Input	Char(10)
9	Message reference key	Input	Char(4)
10	Cursor position option	Input	Char(1)
11	Last list entry	Input	Char(4)
12	Error list entry	Input	Char(4)
13	Wait time	Input	Binary(4)

The Display Panel (QUIDSPP) API displays a panel and waits for the user to press either a function key or the Enter key.

The values for all I/O fields in the panel definition are taken from dialog variables in the variable pool. If the panel contains list areas, these values are also taken from list entries in the lists associated with the open application.

The application program can also control which list entry is initially displayed at the top of a list area by using the Set List Attributes (QUISETLA) API. If the panel contains a list area displaying an incomplete list and if the list does not have enough entries to fill the list area, the UIM calls the program identified by the program dialog variable parameter of the QUISETLA API to add more list entries. The program is called repeatedly until the requested number of entries is added or until the list is marked as complete.

If the panel contains an extended action list area and if the list is not currently active in the open application, the list is activated by the QUIDSPP API.

Any information the user enters into input fields on the panel is validated according to the specifications of the tag language, then saved in the corresponding dialog variables and list entries.

Control returns to the application program only after all necessary functions are completed as defined by UIM processing rules and tag language specifications. The application program can retrieve dialog variables and list entries after the QUIDSPP API returns to determine what values were specified by the user.

The QUIDSPP API can be called with a long or short argument list. For the short argument list, arguments must be passed to the QUIDSPP API with the required parameters described below. For the long argument list, arguments must be passed to the QUIDSPP API with all the required and optional parameters described below.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Function requested
OUTPUT; BINARY(4)
The function requested by the user. Only special UIM-defined functions and functions using the RETURN dialog command are returned to the application program. Any other functions requested by the user are either handled by the UIM or rejected. If they are rejected, an error message is displayed on the panel.

The RETURN dialog command can be specified as follows:

- On the ACTION attribute of the key list item (KEYI), pull-down field choice (PDFLDC), or menu item (MENUI) UIM tags
- On the ENTER and SELECT attributes of the display panel (PANEL) UIM tag

The RETURN dialog command returns numbers from 1 through 32 767. The return values for UIM-defined functions are as follows:

0 The enter function is requested. This value is returned only for a panel with a list area of ACTOR=CALLER, and only when the user has entered options in the list.

-4 The exit function is requested.

-8 The cancel function is requested.

-10 The prompt function is requested. This value is returned only for a panel with an ACTOR=CALLER list area, and only when the user has entered options in the list.

-20 No function was requested before the time specified on the wait time parameter has ended. This value is returned only when the wait time parameter specifies a time-out value.

Panel name
INPUT; CHAR(10)
The name of the panel to display. The panel must be defined in the panel group for the open application.

Redisplay option
INPUT; CHAR(1)
Indicates whether or not the panel is formatted and displayed as a first-time display or as a redisplay.

One of the following values must be used:

Y The panel is formatted as a redisplay. If the panel has not been displayed in the application before, processing for the first-time display is done and this parameter is ignored.

Cursor positioning that is controlled by the program is ignored when the redisplay option is used. Cursor positioning controlled by action list processing is supported when the redisplay option is used.

Dialog variables are edited for a new displayable value only if the variable pool contains a value more current than the value from the last time the panel was displayed. When VARUPD=NO is processed, the panel is redisplayed with values entered in input fields but not stored in the variable pool.

N The panel is formatted as a first-time display. The panel is displayed with the first conditioned-on item at the top of each pageable data and menu area. Pageable information areas are positioned at the first line of text. All dialog variables displayed on the screen are edited to produce a displayable value.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Optional Parameter Group

User task
INPUT; CHAR(1)
Indicates whether or not this panel is the logical start of a new user task. The value used determines whether or not the UIM returns to the application program when the EXIT dialog command is requested by a lower level program, such as an action on the KEYI tag.

The user task has no effect on UIM operations when the EXIT dialog command is specified directly in the definition of this panel as the function key, pull-down choice, or menu item action.

One of the following values must be specified:

N This panel is the logical start of a new user task. If the user generates the EXIT dialog command at a lower call level (for example, by pressing the exit function key on a panel displayed by a list action from the original panel), this panel is redisplayed without returning control to the application program.

O This panel is not the start of a new user task. If the user generates the EXIT dialog command at a lower call level, control returns to the application program with an indication that the exit function was requested. This is the default value when a short argument list is passed to the QUIDSPP API.

Program stack counter
INPUT; BINARY(4)
A number identifying the location in the program stack of the program message queue for the UIM to use. This parameter is used with the program message queue parameter.

Any nonnegative number can be specified for the relative program call number.

0 The message queue of the program specified in the program message queue parameter. This is the default value when a short argument list is passed to the QUIDSPP API.

1 The message queue of the program that calls the program specified in the program message queue parameter is used for messages displayed to the user.

n The message queue of the *n*th program up the stack from the program specified in the program message queue parameter is used for messages displayed to the user.

Program message queue
INPUT; CHAR(10)
The name of the message queue for the UIM to use, or the name of the program to start counting from when using a value other than zero for the program stack counter parameter. The program you specify must be in the program stack. The UIM will:

- Receive a message from this queue to initially display on the message line of the panel. This is done using the value passed for the message reference key parameter.
- Move to this message queue all completion, information, diagnostic, and escape messages sent to the UIM by programs or commands called to process an action defined in the panel. Note that escape messages are changed to diagnostic messages when they are moved.

The following special value can be used:

***CALLER** The application program calling the QUIDSPP API is the starting point for identifying the program message queue. This is the default value when a short argument list is passed to the QUIDSPP API.

Message reference key
INPUT; CHAR(4)
The messages in the program message queue, identified by the program stack counter parameter and the program message queue parameter, that are displayed on the panel. This parameter must be set to the message reference key for the first (oldest) message on the program message queue that should be displayed on the panel. One of the following special values can be used:

(blank) No messages are shown to the user on the initial panel, even if there are messages in the program message queue. This is the default value when a short argument list is passed to the Display Panel (QUIDSPP) API.

FRST All messages in the program message queue, starting with the first message in the queue, are presented on the initial panel.

Cursor position option
INPUT; CHAR(1)
The rules that determine the initial position of the cursor when the panel is displayed. This parameter controls only the initial cursor placement when the panel is displayed by the QUIDSPP API. When a panel is redisplayed by the UIM or by the application program, the cursor remains where it was left by the user.

One of the following values must be used:

A Cursor positioning for action list processing is performed. This value is used when the panel is redisplayed after action list processing when ACTION=CALLER is on the list area (LIST) tag. The cursor and list are positioned as defined by the last list entry and error list entry parameters.

D Default cursor positioning is performed. The specific cursor position depends on whether or not any variables or list entries associated with the panel are marked in error. This is the default value when a short argument list is passed to the display panel (QUIDSPP) API.

P Cursor positioning that is controlled by the program is performed. The cursor is positioned using the current value of the dialog variables associated with the CSRVAR, CSRPOS, CSRLST, and CSREID attributes of the panel definition (PANEL) tag. Cursor positioning controlled by the program is allowed only for a panel containing all four cursor positioning attributes.

With cursor positioning controlled by the program, each pageable, nonlist area is repositioned to make sure that the first input field in error is visible, but the cursor is not positioned at the first input field in error. If the value of any of the dialog variables for cursor position identified in the panel definition is unusable, the cursor defaults to the first input field on the panel.

Last list entry
INPUT; CHAR(4)
The list entry handle for the last action list entry that had an action processed.

This information is used to position the cursor. Cursor positioning for action list processing must be specified in the cursor position option parameter or this parameter is ignored. One of the following special values can be used:

EXTE The last list action processed is for the extended action entry.

NONE The application is providing no value for this parameter. This is the default value when a short argument list is passed to the Display Panel (QUIDSPP) API.

Error list entry
INPUT; CHAR(4)
The list entry handle for the first and only list entry that had an error while processing list actions. This parameter is also set when action list processing is stopped because a user pressed F3 (Exit) or F12 (Cancel). This information displays the correct page of list information.

Cursor positioning for action list processing must be specified in the cursor position option parameter, or this parameter is ignored.

One of the following special values can be used:

EXTE The extended action entry is in error; the UIM does not reposition the list.

NONE The application is providing no value for this parameter. This is the default value when a short argument list is passed to the Display Panel (QUIDSPP) API.

Wait time
INPUT; BINARY(4)
The number of seconds to wait for data to become available from the work station. After this amount of time, the keyboard locks, control returns to the application, and the function requested parameter returns an indication that the wait time ended. Because the wait time ended, input dialog variables associated with the panel are not updated. The number of seconds specified should be a positive integer.

One of the following special values can be used:

-1 The UIM waits indefinitely until data becomes available from the work station. This is the default value when a short argument list is passed to the Display Panel (QUIDSPP) API.

0 The UIM does not wait for data to become available from the work station. The panel is displayed with the keyboard locked, and control returns immediately to the application with an indication in the function requested parameter that the wait time ended.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A13 E Application &3 closed prematurely.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A22 E Panel cannot be displayed in a window.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3E E Application not open for display.
CPF6A3F E &4 was not found in panel group &1.
CPF6A4A E &4 is too large to show as a window.
CPF6A4B E Value for Redisplay parameter not valid.
CPF6A40 E &4 is already in use.
CPF6A41 E CSRPOS(*DFT) required for display &4.
CPF6A42 E Display of panel &4 not allowed.
CPF6A43 E Cannot use program message queue &6.
CPF6A50 E Error was found during display file or print file operation.

Get Dialog Variable (QUIGETV) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Output	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	Error code	I/O	Char(*)

The Get Dialog Variable (QUIGETV) API allows a program to obtain the value of one or more dialog variables by specifying the application program's variable buffer and the name of a variable record defined in the panel group for the open application.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
OUTPUT; CHAR(*)
The program buffer into which dialog variable values are copied. The variable buffer must be large enough to contain all the variables specified in the definition of the variable record. The dialog variables are copied in the order specified in the variable record, which is specified in the variable record name parameter.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the definition of the variable record, which is specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record definition that determines which dialog variables are copied from the application variable pool to the variable buffer. The variable record must be defined in the panel group for the open application.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.

Get List Entry (QUIGETLE) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Output	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Positioning option	Input	Char(4)
7	Copy option	Input	Char(1)
8	Selection criteria	Input	Char(20)
9	Selection handle	Input	Char(4)
10	Extend option	Input	Char(1)
11	List entry handle	Output	Char(4)
12	Error code	I/O	Char(*)

The Get List Entry (QUIGETLE) API accesses an entry in a list and optionally updates the corresponding dialog variables to the values in the list entry.

The entry accessed depends on the positioning option parameter and the value of the current entry pointer for the list. On return to the application program, the current entry pointer in the list points to the entry retrieved, except when retrieving the extended action entry. The extended action entry is a pseudo list entry that can be retrieved using this API, and can be updated using the Update List Entry (QUIUPDLE) API.

If requested, the corresponding dialog variables contain the values from the list entry retrieved.

If an error variable corresponding to a field in the list entry is specified in the variable record, identified by the variable record name parameter, it is set according to the error state of the corresponding dialog variable.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) or Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
OUTPUT; CHAR(*)
The program buffer into which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record definition.

If the variable record name parameter specifies the name of a variable record defined in the panel group for this open application, dialog variables are copied into the variable buffer from the application variable pool after the list entry is retrieved. This parameter does the same job as using the Get Dialog Variable (QUIGETV) API immediately after the QUIGETLE API.

The variable buffer must be large enough to contain all the variables specified in the variable record definition.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the variable record definition, which is specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied between the application variable pool and the variable buffer. The variable record must be defined in the panel group for the open application. The following special value can be used:

***NONE** The QUIGETV API is not done during the QUIGETLE API; the parameter for the variable buffer is ignored when this value is used.

List name
INPUT; CHAR(10)
The name of the list from which an entry is retrieved. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) API or Add List Multiple Entries (QUIADDLM) API, or when the list attributes are set with the Set List Attributes (QUISETLA) API.

Positioning option
INPUT; CHAR(4)
The placement of the current entry pointer to the list entry specified. If a positioning error occurs, the current list position is not changed. One of the following values must be specified to set the entry pointer to the appropriate position:

Value	Pointer Position
BOT	The bottom of the list. A special position just after the last list entry in a list that is complete at the bottom.
FRST	The first entry in the currently built list. If the list is empty, an error is reported.
FSLT	The entire list is searched in a forward direction to locate an entry satisfying the condition specified by the selection criteria parameter. If the list is empty, an error is reported. The current entry pointer for the list entry is repositioned at the first entry in the list and the search proceeds forward, including the first entry. The search does not wrap.
HNDL	The list entry specified by the list entry handle parameter.
LAST	The last entry in the currently built list. If the list is empty, an error is reported.
LSLT	The entire list is searched in a backward direction to locate an entry satisfying the condition specified by the selection criteria parameter. If the list is empty, an error is reported. The current list entry pointer is repositioned to the last entry in the list and the search proceeds backward, including the last entry. The search does not wrap.
NEXT	The current entry pointer is advanced one entry, pointing to the next entry in the list.
NSLT	The list is searched in a forward direction to locate an entry satisfying the condition specified by the selection criteria parameter. The search starts with the entry after the current list entry pointer and does not wrap.
PREV	The current entry pointer is moved back one entry, pointing to the previous entry in the list.
PSLT	The list is searched in a backward direction to locate an entry satisfying the condition specified by the selection criteria parameter. The search starts with the entry before the current list entry pointer and does not wrap.
SAME	The current entry pointer remains unchanged, pointing to the same entry in the list.
TOP	The top of the list. A special position just prior to the first list entry in a list that is complete at the top.

Note: After running the QUIGETLE API once with the positioning option parameter of FSLT or LSLT, the application program should change the positioning option parameter to NSLT or PSLT. It does this to avoid repeatedly positioning at the beginning or end of the list and searching the same list entries.

Copy option
INPUT; CHAR(1)
Determines whether or not the data values in the list entry are copied into the corresponding dialog variables when positioning is complete. One of the following values must be specified:

Y The data values in the current list entry after the positioning operations are performed are copied into corresponding dialog variables. This value must be specified if the variable record name parameter specifies the name of a variable record defined in the panel group for the open application. This value is not allowed when the current entry is positioned to TOP or BOT.

N The data values in the current list entry after positioning the list are not copied into corresponding dialog variables.

Selection criteria
INPUT; CHAR(20)
The selection criteria used when positioning the list by variable selection. Positioning by variable selection allows the application program to search forward to find the next entry, or backward to find a previous entry satisfying a given condition. The search is not dependent on sorted list entries, but the application program might need to build the list in sorted order to get a consistent result when application comparison operators are other than equal (EQ) or not equal (NE).

This parameter is only used when one of the following values is specified for the positioning option parameter: FSLT, LSLT, NSLT, or PSLT. It is ignored if any other value is specified.

The relational condition defined by this parameter must exist between the value of a dialog variable and that dialog variable's corresponding value in a list entry when selection positioning is performed.

The first 10 characters of this parameter must contain a comparison operator. The operator must be left-adjusted and padded with blanks.

One of the following values must be specified for the comparison operator:

EQ The list entry value equals the dialog variable value.

NE The list entry value is not equal to the dialog variable value.

GT The list entry value is greater than the dialog variable value.

LT The list entry value is less than the dialog variable value.

GE The list entry value is greater than or equal to the dialog variable value.

LE The list entry value is less than or equal to the dialog variable value.

The second 10 characters of this parameter must contain the name of the dialog variable used in the comparison. The dialog variable name specified must be defined in the list being searched.

Selection handle
INPUT; CHAR(4)
The list entry handle used when positioning the current entry pointer to a specific entry. This parameter is used only when the positioning option parameter has the value HNDL; it is ignored if any other value is specified.

The following special value can be specified:

EXTE Retrieves the contents of the extended action entry. The contents of the extended action entry are copied to the associated dialog variables in the variable pool. However, the current entry pointer for the list is not changed when the extended action entry is retrieved.

A list entry handle uniquely distinguishes an entry until it is removed from the list, even if other entries are inserted and removed from the list.

Extend option
INPUT; CHAR(1)
Indicates whether or not an incomplete list is automatically extended in the attempt to retrieve the requested list entry. The list can be extended when one of the following values is specified for the positioning option parameter: NEXT, PREV, TOP, BOT, NSLT, PSLT, FSLT, or LSLT. This parameter is ignored and the list is not extended if any other value is specified for the positioning option parameter.

One of the following values must be specified:

Y The list is extended, if necessary, to find the requested list entry.

N The list is not extended to find the requested list entry. This option can be used if the program calling the QUIGETLE API needs to process only entries already in the list.

List entry handle
OUTPUT; CHAR(4)
The list entry handle from the current entry pointer in the list. This value is the handle of the entry to which the list was last positioned by this API. The following special values may be returned:

TOP The current entry pointer is positioned at the top of the list.

BOT The current entry pointer is set at the bottom of the list.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A2C E Value for Option parameter not valid.
CPF6A2D E Value for Selection Criteria parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A27 E Value for Extend Option parameter not valid.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF6A93 E Operation not valid when current entry is &5.
CPF6A95 E List &4 either not complete or not extended.
CPF6A96 E Variable &5 is not in list definition.
CPF6A97 E Variable &5 not valid type for select comparison.
CPF6A98 E Entry not found in list &4 in panel group &1 in &2.

Get List Multiple Entries (QUIGETLM) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Output	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Positioning option	Input	Char(4)
7	Copy option	Input	Char(1)
8	Selection criteria	Input	Char(20)
9	Selection handle	Input	Char(4)
10	Extend option	Input	Char(1)
11	List entry handle	Output	Char(4)
12	Number of records	Input	Binary(4)
13	Record size	Input	Binary(4)
14	Record count	Output	Binary(4)
15	Error code	I/O	Char(*)

The Get List Multiple Entries (QUIGETLM) API accesses one or more entries in a list and updates the corresponding dialog variables with the values contained in the list entry.

The entry accessed depends on the positioning option parameter and the value of the current entry pointer for

the list. On return to the application program, the current entry pointer in the list points to the last entry retrieved, except when retrieving the extended action entry. The extended action entry can be retrieved using this API, and can be updated using the Update List Entry (QUIUPDLE) API.

When the operation completes successfully, the corresponding dialog variables contain the values from the last list entry retrieved.

If an error variable corresponding to a field in the list entry is specified in the variable record, which is identified by the variable record name parameter, it is set according to the error state of the corresponding dialog variable.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
OUTPUT; CHAR(*)
The program buffer into which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record definition.

Dialog variables are copied into the variable buffer from the application variable pool after the list entry is retrieved. This parameter does the same job as using the Get Dialog Variable (QUIGETV) API immediately after the list entry is retrieved.

The variable buffer must be large enough to contain all the variables specified in the variable record definition.

When the number of records parameter is greater than 1, the size of the variable buffer must be at least equal to the value specified on the number of records parameter multiplied by the value specified on the record size parameter.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the variable record definition specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied between the application variable pool and the variable buffer. The variable record must be defined in the panel group for the open application.

The special value of *NONE may not be used with the QUIGETLM API.

List name
INPUT; CHAR(10)
The name of the list from which an entry is retrieved. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) or Add List Multiple Entries (QUIADDLM) API, or when the list attributes are set with the Set List Attributes (QUISETLA) API.

Positioning option
INPUT; CHAR(4)
Sets the current entry pointer to the list entry specified. If a positioning error occurs, the current list position is not changed. One of the following values must be specified to set the entry pointer to the appropriate position:

Value	Pointer Position
BOT	The bottom of the list. A special position just after the last list entry in a list that is complete at the bottom.
FRST	The first entry in the currently built list. If the list is empty, an error is reported.
FSLT	The entire list is searched in a forward direction to locate an entry satisfying the condition specified by the selection criteria parameter. If the list is empty, an error is reported. The current pointer for the list entry is repositioned at the first entry in the list, and the search proceeds forward, including the first entry. The search does not wrap.
HNDL	The list entry specified by the list entry handle parameter.
LAST	The last entry in the currently built list. If the list is empty, an error is reported.
LSLT	The entire list is searched in a backward direction to locate an entry satisfying the condition specified by the selection criteria parameter. If the list is empty, an error is reported. The current list entry pointer is repositioned to the last entry in the list, and the search proceeds backward, including the last entry. The search does not wrap.
NEXT	The current entry pointer is advanced one entry, pointing to the next entry in the list.
NSLT	The list is searched in a forward direction to locate an entry satisfying the condition specified by the selection criteria parameter. The search starts with the entry after the current list entry pointer, and does not wrap.
PREV	The current entry pointer is moved back one entry, pointing to the previous entry in the list.

PSLT The list is searched in a backward direction to locate an entry satisfying the condition specified by the selection criteria parameter. The search starts with the entry before the current list entry pointer and does not wrap.

SAME The current entry pointer remains unchanged, pointing to the same entry in the list.

TOP The top of the list. A special position just prior to the first list entry in a list that is complete at the top.

Note: After running the QUIGETLM API once with the positioning option parameter of FSLT or LSLT, the application program should change the positioning option parameter to NSLT or PSLT. It does this to avoid repeatedly positioning at the beginning or end of the list and searching the same list entries.

Copy option
INPUT; CHAR(1)
Determines whether or not the data values in the list entry are copied into the corresponding dialog variables when positioning is complete. One of the following values must be specified:

Y The data values in the current list entry after the positioning operations are performed are copied into corresponding dialog variables. This value must be specified if the variable record name parameter specifies the name of a variable record defined in the panel group for the open application. Y must be used if the number of records parameter is greater than 1. This value is not allowed when the current entry is positioned to TOP or BOT.

N The data values in the current list entry after positioning the list are not copied into corresponding dialog variables.

Selection criteria
INPUT; CHAR(20)
The selection criteria used when positioning the list by variable selection. Positioning by variable selection allows the application program to search forward to find the next entry, or backward to find a previous entry satisfying a given condition. The search is not dependent on sorted list entries, but the application program might need to build the list in sorted order to get a consistent result when using comparison operators other than equal (EQ) or not equal (NE).

This parameter is only used when FSLT, LSLT, NSLT, or PSLT is specified for the positioning option parameter. It is ignored if any other value is specified.

The relational condition defined by this parameter must exist between the dialog variable value and that dialog variable's corresponding value in a list entry when selection positioning is performed.

The first 10 characters of this parameter must contain a comparison operator. The operator must be left-adjusted and padded with blanks.

One of the following values must be specified for the comparison operator:

EQ The list entry value equals the dialog variable value.

NE The list entry value is not equal to the dialog variable value.

GT The list entry value is greater than the dialog variable value.

LT The list entry value is less than the dialog variable value.

GE The list entry value is greater than or equal to the dialog variable value.

LE The list entry value is less than or equal to the dialog variable value.

The second 10 characters of this parameter contain the name of the dialog variable used in the comparison. The dialog variable name specified must be defined in the list being searched.

Selection handle
INPUT; CHAR(4)
The list entry handle used when positioning the current entry pointer to a specific entry. This parameter is used only when the positioning option parameter has the value HNDL; it is ignored if any other value is specified.

The following special value can be specified:

EXTE Retrieves the content of the extended action entry. The contents of the extended action entry are copied to the associated dialog variables in the variable pool. However, the current entry pointer for the list is not changed when the extended action entry is retrieved.

A list entry handle uniquely distinguishes an entry until it is removed from the list, even if other entries are inserted and removed from the list.

Extend option
INPUT; CHAR(1)
Indicates whether or not an incomplete list is automatically extended in the attempt to retrieve the requested list entry. The list can be extended when one of the following values is specified for the positioning option parameter: NEXT, PREV, TOP, BOT, NSLT, PSLT, FSLT, or LSLT. If any other value is specified for the positioning option parameter, this parameter is ignored and the list is not extended.

One of the following values must be specified:

Y The list is extended, if necessary, to find the requested list entry.

N The list is not extended to find the requested list entry. This option can be used if the program calling the QUIGETLE API needs to process only entries already in the list.

List entry handle
OUTPUT; CHAR(4)
The list entry handle from the current entry pointer in the list. This value is the handle of the entry to which the list was last positioned by this API.

The following special values may be returned.

TOP The current entry pointer is positioned at the top of the list.
BOT The current entry pointer is set at the bottom of the list.

Number of records
INPUT; BINARY(4)
The total number of entries to retrieve from the list. The following special value can be used:

1 Only one entry is retrieved from the list. The record size and record count parameters are ignored when this value is used.

When this parameter is greater than 1, the variable buffer must contain space for all the entries retrieved from the list.

When this parameter is greater than 1 (multiple entries are being retrieved), the positioning option parameter must have one of the following values: NEXT, PREV, NSLT, PSLT, FSLT, or LSLT.

Record size
INPUT; BINARY(4)
The size of each record within the variable buffer when multiple records are retrieved from the list. This parameter also calculates the offset of each record after the first one.

When the number of records parameter is 1, this parameter is ignored.

Record count
OUTPUT; BINARY(4)
The number of list entries actually retrieved. When the number of records parameter is 1, this parameter is ignored.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A06 E Record size or record number too large.
CPF6A13 E Application &3 closed prematurely.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A2C E Value for Option parameter not valid.
CPF6A2D E Value for Selection Criteria parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A27 E Value for Extend Option parameter not valid.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF6A93 E Operation not valid when current entry is &5.
CPF6A95 E List &4 either not complete or not extended.
CPF6A96 E Variable &5 is not in list definition.
CPF6A97 E Variable &5 not valid type for select comparison.
CPF6A98 E Entry not found in list &4 in panel group &1 in &2.

Open Display Application (QUIOPNDA) API

Parameters

Required Parameter Group:

1	Application handle	Output	Char(8)
2	Qualified panel group name	Input	Char(20)
3	Application scope	Input	Binary(4)
4	Exit parameter interface	Input	Binary(4)
5	Full-screen help	Input	Char(1)
6	Error code	I/O	Char(*)

The Open Display Application (QUIOPNDA) API initiates a UIM display application by opening the panel group that the application program specifies. The QUIOPNDA API and the Close Application (QUICLOA) API must be used in pairs to open and close each UIM display application.

Multiple applications can be opened at the same time. Each open application contains a complete set of dialog variables and active lists, and is independent of other open applications. A panel group can be opened more than once per job, but each call of the QUIOPNDA API initiates a new UIM display application and returns a unique application handle.

Authorities and Locks

Library Authority *READ
Panel Group Authority *USE
Panel Group Lock *SHRRD

Required Parameter Group

Application handle
OUTPUT; CHAR(8)
The application handle for the opened application. The QUIOPNDA API returns a unique handle for each application opened. This handle must be provided as a parameter to every other UIM API that operates on the application, including the QUICLOA API, which must be used to close the application.

Qualified panel group name
INPUT; CHAR(20)
The name of the *PNLGRP object opened for this UIM application. The first 10 characters contain the name of the *PNLGRP object, and the second 10 characters contain the name of the library in which the panel group resides. These special values can be used for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Application scope
INPUT; BINARY(4)
The scope of the resources for the application. The UIM uses the scope to determine whether or not to automatically close the application when reclaim resource processing is performed through the Reclaim Resource (RCLRSC) and the End Request (ENDRQS) commands. During reclaim resource processing, the UIM closes all applications whose scope is no longer active.

One of the following values must be used:

-1 The calling program is the scope for the application. The UIM automatically closes the application during reclaim resource processing if the application program is no longer active.

0 The job is the scope for the application. The application is not automatically closed by the UIM until the job is ended.

Exit parameter interface
INPUT; BINARY(4)
The type of parameter interface used with exit programs and programs called as a result of the CALL dialog command for the UIM application being opened.

All exit and CALL programs receive a single parameter or multiple parameters, which are space pointers to information describing the state of the UIM application.

One of the following values must be used:

0 Used by programs written in languages that can efficiently process structures.

1 Used by programs written in languages that cannot efficiently process structures. This value indicates that all application programs called as exits or as a result of the CALL dialog command accept the parameter lists described for interface level 1.

2 Used by programs written in languages that cannot efficiently process structures. This value indicates that all application programs called as exits or as a result of the CALL dialog command accept the parameter lists described for interface level 2.

For a detailed description of the structure passed for each type of exit and CALL program and for a description of the exit parameter lists, see Chapter 29, "User Interface Management Exit Programs" on page 29-1.

Full-screen help
INPUT; CHAR(1)
Determines whether or not the UIM help for the application is displayed in pop-up windows or with a full screen. One of the following values must be used:

Y The online help information is displayed with a full screen.

N The online help information is displayed using pop-up windows, unless the user profile indicates that help is displayed with a full screen.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A1E E Object cannot be used with this device or print file.
CPF6A12 E Unable to open panel group.
CPF6A17 E Panel group &1 in library &2 is not at the current release level.
CPF6A2A E Value for Application Scope parameter not valid.
CPF6A2E E Value for Exit Program Interface parameter not valid.
CPF6A2F E Value for Full Screen Help parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A26 E Resources not available to open application.

Open Print Application (QUIOPNPA) API

Parameters

Required Parameter Group:

1	Application handle	Output	Char(8)
2	Qualified panel group name	Input	Char(20)
3	Application scope	Input	Binary(4)
4	Exit parameter interface	Input	Binary(4)
5	Qualified name of printer device file	Input	Char(20)
6	Alternative file name	Input	Char(10)
7	Share open data path	Input	Char(1)
8	User data	Input	Char(10)
9	Error code	I/O	Char(*)

The Open Print Application (QUIOPNPA) API initiates a UIM print application by opening the panel group that the application program specifies. The QUIOPNPA API and the Close Application (QUICLOA) API must be used in pairs to open and close each UIM print application.

Multiple applications can be opened at the same time. Each open application contains a complete set of dialog

variables and active lists, and is independent of other open applications. The same panel group can be opened more than once per job, but each call of the QUIOPNPA API initiates a new UIM print application and returns a unique application handle.

Authorities and Locks

Library Authority	*READ
Panel Group Authority	*USE
Panel Group Lock	*SHRRD
Printer Device File Authority	*USE
Printer Device File Lock	*SHRNUP

Required Parameter Group

Application handle
OUTPUT; CHAR(8)
The application handle for the open application. The Open Display Application (QUIOPNDA) API returns a unique handle for each application opened. This handle must be provided as a parameter to every other UIM API that operates on the application, including the QUICLOA API, which must be used to close the application.

Qualified panel group name
INPUT; CHAR(20)
The name of the *PNLGRP object opened for this UIM application. The first 10 characters contain the name of the *PNLGRP object, and the second 10 characters contain the name of the library in which the panel group resides. These special values can be used for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Application scope
INPUT; BINARY(4)
The scope of the resources for the application. The UIM uses the scope to determine whether or not to automatically close the application when reclaim resource processing is performed through the Reclaim Resources (RCLRSC) and the End Request (ENDRQS) commands. During reclaim resource processing, the UIM closes all applications whose scope is no longer active.

One of the following values must be used:

-1 The calling program is the scope for the application. The UIM closes the application during reclaim resource processing if the application program is no longer active.

0 The job is the scope for the application. The application is not automatically closed by the UIM until the job is ended.

Exit parameter interface
INPUT; BINARY(4)
The type of parameter interface used with exit programs and programs called as a result of the CALL dialog command for the UIM application being opened.

All exit and CALL programs receive a single parameter or multiple parameters, which are space pointers to information describing the state of the UIM application.

One of the following values must be used:

0 Used by programs written in languages that can efficiently process structures.

1 Used by programs written in languages that cannot efficiently process structures. This value indicates that all application programs called as exits or as a result of the CALL dialog command accept the parameter lists described for interface level 1.

2 Used by programs written in languages that cannot efficiently process structures. This value indicates that all application programs called as exits or as a result of the CALL dialog command accept the parameter lists described for interface level 2.

For a detailed description of the exit parameter lists, and for a description of the type of structure passed for each type of exit and CALL program, see Chapter 29, "User Interface Management Exit Programs" on page 29-1.

Qualified name of printer device file
INPUT; CHAR(20)
The name of the printer device file used in print operations. The first 10 characters contain the *FILE object name, and the second 10 characters contain the name of the library in which the printer device file resides. These special values can be used for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Alternative file name
INPUT; CHAR(10)
An alternative name for the spooled output file. The following special value can be used:

***NONE** There is no alternative name for the spooled output file. The name of the spooled output file is the name of the printer device file.

Share open data path
INPUT; CHAR(1)
Determines whether or not the printer device file's open data path (ODP) is shared. Sharing the ODP allows multiple UIM applications to print to the same spooled output file. One of the following values must be used:

Y The ODP is shared.
N The ODP is not shared.

User data
INPUT; CHAR(10)
An attribute of the spooled output file. Typically, this data indicates what command produced the spooled output. The following special value can be used:

***NONE** There is no user data associated with the spooled output file.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A1C E Unable to add print function.
CPF6A1E E Object cannot be used with this device or print file.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A12 E Unable to open panel group.
CPF6A17 E Panel group &1 in library &2 is not at the current release level.
CPF6A2A E Value for Application Scope parameter not valid.
CPF6A2E E Value for Exit Program Interface parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A26 E Resources not available to open application.
CPF9850 E Override of printer file &1 not allowed.

Print Panel (QUIPRTP) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Print panel name	Input	Char(10)
3	Eject option	Input	Char(1)
4	Error code	I/O	Char(*)

The Print Panel (QUIPRTP) API prints a panel to the printer file for an opened print application. The values for all output fields used in the panel definition are taken from dialog variables in the variable pool. If the panel contains list areas, the values are also taken from list entries in the lists associated with the open application.

If the panel contains a list area that is incomplete at the bottom, the UIM automatically calls the program identified by the program dialog variable parameter of the Set List Attributes (QUISETLA) API to acquire more list entries. The program is called repeatedly until either the requested number of entries is added to the list or the list is marked complete at the bottom. For lists that are incomplete at the top, printing begins with the first entry in the list.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Print panel name
INPUT; CHAR(10)
The name of the print head panel or print panel defined in the panel group for the open application.

Eject option
INPUT; CHAR(1)
Determines whether or not this panel begins on a new page. An automatic page eject is done when a print head panel is printed after a print panel. However, even if Y is specified on the QUIPRTP API when the next print panel is printed, it does not cause a second page eject.

One of the following values must be used:

Y The panel is printed at the top of a new page.
N The panel is not always printed at the top of a new page.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A1F E An active display already exists for this application.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A13 E Application &3 closed prematurely.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A18 E Print heading must be specified first.
CPF6A19 E Prologue is only allowed in first heading.
CPF6A23 E Page length too small to print the list column headings.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3B E Application not open for print.
CPF6A3E E Application not open for display.
CPF6A3F E &4 was not found in panel group &1.
CPF6A50 E Error was found during display file or print file operation.

Put Dialog Variable (QUIPUTV) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Input	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	Error code	I/O	Char(*)

The Put Dialog Variable (QUIPUTV) API updates the value of one or more dialog variables by specifying variable buffer of the application program and naming the variable record defined in the panel group for the open application.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
INPUT; CHAR(*)
The program buffer from which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record, which is named in the variable record name parameter.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the definition of the variable record, which is specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied from the variable buffer into the application variable pool. The variable record must be defined in the panel group for the open application.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A36 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A37 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.

Remove List Entry (QUIRMVLE) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	List name	Input	Char(10)
3	Extend option	Input	Char(1)
4	List entry handle	Output	Char(4)
5	Error code	I/O	Char(*)

The Remove List Entry (QUIRMVLE) API removes the list entry identified by the value of the current entry pointer for the list. The current entry pointer is always updated to the entry before the one removed. If the first list entry is removed, the current entry pointer is set to the top of the list if the list is complete at the top.

If the list is incomplete at the top, the UIM calls the incomplete list extension program to add another entry to the list.

If the list entry identified by the display position attribute parameter of the Set List Attributes (QUISETLA) API is removed, the display position attribute is set at the entry before the one that was removed. If the new display position attribute is the first entry in the list, the display position attribute is set at the top of the list (logically the entry before the first entry in the list) if the list is complete at the top.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

List name
INPUT; CHAR(10)
The name of the list from which an entry is removed. If the list is not currently active in the open application, an error message is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) API, Add List Multiple Entries (QUIADDLM) API, or its attributes are set with the QUISETLA API.

Extend option
INPUT; CHAR(1)
Specifies whether or not an incomplete list is automatically extended in an attempt to remove the first entry from a list that is incomplete at the top. One of the following values must be specified:

Y The list is extended, if necessary, to add at least one entry to the top of the list or to mark the list as complete at the top.

N The list is not extended to find the entry before the entry being removed, and the entry is not removed from the list.

List entry handle
OUTPUT; CHAR(4)
The list entry handle value from the current entry pointer. A list entry handle uniquely distinguishes an entry until it is removed from the list, even if other entries are inserted and removed from the list. A value of TOP indicates that the current entry pointer is positioned at the top of the list.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A13 E Application &3 closed prematurely.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A27 E Value for Extend Option parameter not valid.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF6A93 E Operation not valid when current entry is &5.
CPF6A94 E Incomplete list &4 requires extension.
CPF6A95 E List &4 either not complete or not extended.

Remove Pop-Up Window (QUIRMVPW) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Remove option	Input	Char(1)
3	Error code	I/O	Char(*)

The Remove Pop-Up Window (QUIRMVPW) API removes the pop-up window created by the Add Pop-Up Window (QUIADDPW) API. After calling the QUIRMVPW API, subsequent calls to the Display Panel (QUIDSPP) API display either a full-screen panel or a panel in a pop-up window defined by a previous call of the QUIADDPW API. A pop-up window cannot be removed if an action or exit program is currently being processed for a panel displayed in that pop-up window.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Remove option
INPUT; CHAR(1)
The pop-up windows that are removed from the open application. One of the following values must be used:

I All inactive pop-up windows. A pop-up window is inactive if there are no panels currently displayed in that window.

L The most recently added pop-up window.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3E E Application not open for display.
CPF6A84 E Window cannot be removed at this time.
CPF6A90 E Value not correct. Reason code &3.

Remove Print Application (QUIRMVPA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Close option	Input	Char(1)
3	Error code	I/O	Char(*)

The Remove Print Application (QUIRMVPA) API stops print functions in a previously opened display application. The Add Print Application (QUIADDPA) API and the QUIRMVPA API are used in pairs to add and remove printing from display applications.

The QUIRMVPA API closes the printer file for the application. If the QUIRMVPA API is not performed before the application is closed, the Close Application (QUICLOA) API performs the QUIRMVPA API.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Close option
INPUT; CHAR(1)
Specifies whether or not to perform a normal or abnormal close operation on the printer file. One of the following values must be specified:

A An abnormal close operation is performed, and the trailer message is not printed.

M A normal close operation is performed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A1B E Application does not have an open print file.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.

Retrieve List Attributes (QUIRTVLA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	List name	Input	Char(10)
3	Receiver	Output	Char(*)
4	Receiver length	Input	Binary(4)
5	Error code	I/O	Char(*)

The Retrieve List Attributes (QUIRTVLA) API retrieves the following list attributes:

- The list contents attribute, indicating whether or not all entries are present in the list, and which entries are missing if it is incomplete
- The name of the dialog variable that identifies the program called when the UIM needs to add entries to an incomplete list
- The display position attribute, which is the list entry handle for the entry presented at the top-most row of any list area that displays a list
- The allow trim attribute, which indicates whether or not the UIM trims a full list when adding new entries

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or by the Open Print Application (QUIOPNPA) API when the application is opened.

List name
INPUT; CHAR(10)
The name of the list whose attributes are retrieved. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) or Add List Multiple Entries (QUIADDLM) API, or the fist time the list's attributes are set with the Set List Attributes (QUISETLA) API.

Receiver
OUTPUT; CHAR(*)
The current attributes of the list. For the format of the receiver variable, see "Format of Data Returned."

Receiver length
INPUT; BINARY(4)
The amount of data the application program is prepared to receive. If the length specified is larger than the amount of data available, the receiver is not changed beyond the amount of data available.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of Data Returned

The format of the data available, returned in the receiver parameter, is as follows:

CHAR(4) The list contents attribute, indicating whether or not the list contains all or some of the entries available for display or printing. For more information about the meaning of each possible return value, see the "Set List Attributes (QUISETLA) API" on page 28-23. The following values can be returned in this parameter:

ALL The list is complete. This value is returned when either the QUISETLA API has not been called to set the list attribute or when ALL is the last list contents attribute specified on a call to the QUISETLA API for the list.

TOP Only the top part of an incomplete list is available. This value is returned when TOP is the last list contents attribute specified on a call to the QUISETLA API for the list.

BOT Only the bottom part of an incomplete list is available. This value is returned when BOT is the last list contents attribute specified on a call to the QUISETLA API for the list.

MORE Only the middle part of an incomplete list is available. This value is returned when MORE is the last list contents attribute specified on a call to the QUISETLA API for the list.

CHAR(10) The name of the dialog variable identifying the program the UIM calls when more entries are needed in an incomplete list.

CHAR(4) The display position attribute, which is the list entry handle at the top of the list area on the next panel that displays this list. The UIM does not use the display position attribute for print applications.

The value returned is the handle for a specific entry in the list, or one of the following special values:

TOP The top entries in the list are displayed.

BOT The bottom entries in the list are displayed.

If the list entry identified by the display position attribute of a list is removed before the list is displayed on a panel, the UIM adjusts the attribute to display the entry before the one that was removed. The value returned by this API is not meaningful if entries are removed from the list after the display position attribute is retrieved.

CHAR(1) The allow trim attribute, which indicates whether or not the UIM trims the list when a new list entry causes the list to exceed its maximum size. For additional details, see the "Set List Attributes (QUISETLA) API."

One of the following values is returned:

Y The UIM automatically trims the list.

N The UIM does not automatically trim the list.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.

Set List Attributes (QUISETLA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	List name	Input	Char(10)
3	List contents	Input	Char(4)
4	Program dialog variable	Input	Char(10)
5	Display position attribute	Input	Char(4)
6	Allow trim attribute	Input	Char(1)
7	Error code	I/O	Char(*)

The Set List Attributes (QUISETLA) API sets the following list attributes:

- The list contents attribute, which indicates whether or not all entries are present in the list and which entries are missing if it is incomplete
- The name of the dialog variable that identifies the program the UIM calls when additional entries are needed in an incomplete list
- The display position attribute, which is the list entry handle for the entry presented at the top row of a list area displaying the list
- An attribute indicating whether or not the UIM trims the list if the maximum list size is exceeded when adding a new entry to the list

If the QUISETLA API has not been called for a list since the first entry was inserted, the list contents default value is ALL and the list is complete. An error message is displayed if the user attempts to page beyond either end of the list. The UIM does not call a program for more entries or return to the program that called the Display Panel (QUIDSPP) API.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or by the Open Print Application (QUIOPNPA) API when the application is opened.

List name
INPUT; CHAR(10)
The name of the list whose attributes are changed. If the list is not currently active in the open application, it is activated by this API.

List contents
INPUT; CHAR(4)
Indicates whether or not the list contains all or some of the entries available. One of the following values must be specified:

SAME This list contents attribute of the list is not changed.

ALL The entire list has been built. A message is displayed if the user attempts to page beyond the top or the bottom of the list.

TOP Only the top part of the list is built. A message is displayed if the user attempts to page beyond the beginning of the list. If the user attempts to page beyond the bottom of the list, the program specified by the program dialog variable parameter is called by the UIM, with parameters indicating the need for more entries at the bottom of the list.

BOT Only the bottom part of the list is built. A message is displayed if the user attempts to page beyond the bottom of the list. If the user attempts to page beyond the top of the list, the program specified by the program dialog variable parameter is called by the UIM, with parameters indicating the need for more entries at the top of the list.

MORE Only the middle of the list is built. If the user attempts to page beyond either the top or the bottom the list, the program specified by the program dialog variable parameter is called by the UIM, with parameters indicating the need for more entries at the top or bottom of the list.

If the application program marks the list complete in a given direction, the application program cannot subsequently mark the list incomplete in that same direction.

Program dialog variable
INPUT; CHAR(10)
The name of a dialog variable in an open application. This dialog variable contains information identifying the program called by the UIM when more entries must be added to an incomplete list. The dialog variable must be defined and set properly according to the rules used for the CALL dialog command.

For a description of the CALL dialog command, see the *Guide to Programming Displays*. For a description of the interface between the UIM and the incomplete list exit program, see Chapter 29, "User Interface Management Exit Programs" on page 29-1.

The following special value can be used:

***SAME** The program dialog variable is not changed.

Display position attribute
INPUT; CHAR(4)
The list entry that should be positioned at the top of the list area on the next panel that displays this list. The UIM does not use the display position attribute when printing the list.

The value specified must be the list entry handle currently existing in the list or one of the following special values:

SAME The display position attribute of the list is not changed.

TOP The entries at the top of the list should be positioned at the top of the list area on the next panel.

BOT The bottom *n* entries of the list should be displayed in the list area on the next panel, where *n* is the number of list entries that can be displayed in the current view of the list. If the list contains fewer than *n* list entries, the complete list is displayed.

If the list entry identified by the display position attribute of a list is removed before the list is displayed on a panel, the UIM automatically adjusts the attribute to display the entry before the one removed. If there is no previous entry in the list, the attribute is set at the top of the list, and the next panel displaying the list shows the first entry in the list.

During incomplete list extension, the application program must specify SAME for the display position attribute.

Allow trim attribute
INPUT; CHAR(1)
Indicates whether or not the UIM should trim the list during the Add List Entry (QUIADDLE) or the Add List Multiple Entries (QUIADDLM) API if adding the new entry exceeds the maximum list size. The UIM trims only lists that have been marked as incomplete by using the QUISETLA API. When the UIM removes the list entry, the UIM marks the list incomplete in the direction from which the entry was removed, regardless of whether or not the list was previously incomplete in this direction.

If an application is adding an entry to the bottom of the list when the maximum size list is reached, the UIM removes the first list entry and marks the list incomplete at the top. The incomplete list exit program of the application must be prepared to add list entries in either direction of an incomplete list.

When the incomplete list exit program is called to add list entries, the QUIADDLE or QUIADDLM API can be called to add entries in the middle of the list. If the entries are added to the middle of the list when the list is complete; trimming does not take place and an error is reported indicating that the maximum list size has been reached.

One of the following values must be specified:

S The allow trim attribute of the list should not be changed.

Y The UIM automatically trims the list.

N The UIM does not automatically trim the list.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.

CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A9A E Dialog variable required to specify incomplete list.
CPF6A9B E Dialog variable &5 not a pointer variable.
CPF6A9C E List entry identifier not correct.
CPF6A9E E Value for Allow Trim parameter not valid.
CPF6A9F E Attribute for list &4 not valid.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A99 E Dialog variable &5 not found in panel group &1 in &2.

Set Screen Image (QUISETSC) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Error code	I/O	Char(*)

The Set Screen Image (QUISETSC) API establishes the screen image for a UIM application. This API is used when a pop-up window is displayed over a panel that was not displayed using the same UIM application.

For displaying pop-up windows over data description specifications (DDS) screens, it is the responsibility of the application program to save and restore the display before and after displaying the pop-up window. This can be done by specifying RSTDSP(*YES) for the DDS display file.

When the QUISETSC API is called, a read screen command is sent to the requester device. The screen image returned from the read screen operation is saved for the open application. Subsequent panels displayed in pop-up windows are displayed on top of this image. Using this API to set the screen image is functionally similar to using the Display Panel (QUIDSPP) API to establish the primary panel on which pop-up windows are overlaid.

The UIM does not preserve the extended character buffer (ECB) attributes on the underlying panel. This means that the UIM issues a read screen command instead of a read screen with ECB command. Any highlighting on the underlying panel that was specified using ECB attributes is lost. When the application redisplays the underlying panel after the pop-up window is removed, the correct highlighting is restored.

No half-index ECB attributes are preserved.

Any double-byte character string (DBCS) data in the underlying panel is supported, including DBCS data whose shift-in and shift-out characters are specified in ECB attributes.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A1F E An active display already exists for this application.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3E E Application not open for display.
CPF6A50 E Error was found during display file or print file operation.

Update List Entry (QUIUPDLE) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Input	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Option	Input	Char(4)
7	List entry handle	Output	Char(4)
8	Error code	I/O	Char(*)

The Update List Entry (QUIUPDLE) API updates the list entry identified by the current entry pointer for the list or the extended action entry. The current contents of all dialog variables corresponding to dialog variables in the list are saved in the entry. The current entry pointer of the list is not changed by this operation.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
INPUT; CHAR(*)
The program buffer from which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record definition.

If the variable record name parameter specifies the name of a variable record, which is defined in the panel group for this open application, dialog variables are copied from the variable buffer to the application variable pool before the list entry is updated. The operation of this parameter is the same as using the Put Dialog Variable (QUIPUTV) API immediately before the QUIUPDLE API.

The variable buffer must be large enough to contain all the variables specified in the variable record definition.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer provided. The buffer must be large enough to contain all the dialog variables in the definition of the variable record, specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied between the variable buffer and the application variable pool. The variable record must be defined in the panel group for the open application.

The following special value can be used:

***NONE** The QUIPUTV API is not done during the QUIUPDLE API. The variable buffer parameter is ignored when this value is used.

List name
INPUT; CHAR(10)
The name of the list in which an entry is updated. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) API or the Add List Multiple Entries (QUIADLM) API, or its attributes are set with the Set List Attributes (QUISETLA) API.

Option
INPUT; CHAR(4)
The updated list entry. One of the following values must be specified:

EXTE The extended action entry is updated.

SAME The list entry identified by the current entry pointer is updated.

List entry handle
OUTPUT; CHAR(4)
The list entry handle for the updated list entry. A list entry handle uniquely distinguishes an entry until it is removed from the list, even if other entries are inserted and removed from the list.

When the option parameter has the value EXTE, this parameter returns the value EXTE, indicating that the extended action entry is updated.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA1 E The value of the action field is not correct at this time. Reason code &5.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A28 E Value for Option parameter not valid.
CPF6A36 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A37 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF6A93 E Operation not valid when current entry is &5.

Chapter 29. User Interface Management Exit Programs

Through panel definitions and APIs, an application program can cause the UIM to call other application programs as part of normal panel management. These programs are divided into two classes: programs called through the CALL dialog command and programs called as exit programs.

This chapter describes the different exits and CALL situations. The UIM does not require a program to follow prescribed actions in its operating environment; attempts to use these exit programs for purposes other than the intended purpose may cause problems for applications and their users.

The CALL dialog command is provided so that an application can link programs and panels without a command interface.

The CALL dialog command causes program calls to handle one of the following requests:

- Function keys
- Menu items
- Action list options
- Pull-down field choices

Exit programs allow applications to perform functions for which the UIM does not provide generic support. Exit program capability exists for the following purposes:

- General panel checking
- Action list update processing
- Incomplete list processing
- Application formatting of data
- Cursor-sensitive prompting

Calling an Exit Program

All programs are specified by naming a dialog variable, which identifies the program to call. For the incomplete list exit program, the dialog variable name is specified on the call to the Set List Attributes (QUISETLA) API. For all other exit and CALL programs, the dialog variable name is specified as a parameter to the CALL dialog command in the tag language source.

The UIM uses the current value of the dialog variable to identify the program to call. Depending on the definition of the dialog variable, there are three different ways to call the program:

Call by address
When the dialog variable specified is defined with BASETYPE=PTR on the class definition (CLASS) tag, the program is called by address.

It is the responsibility of the application program to ensure that the pointer is set correctly.

Call by name
When the dialog variable specified is defined with BASETYPE='CHAR 20' on the CLASS tag, the program is called by name. It is the responsibility of the application program to ensure that the variable contains the object and library name of the program called.

Extended program model (EPM) call
When the dialog variable specified is defined with BASETYPE='CHAR 130' on the CLASS tag, the program is called in the extended program model (EPM) environment. This type of call is used when the target program is written in an EPM language such as Pascal or C/400. It is the responsibility of the application program to ensure that the variable contains the appropriate information for calling an entry point for an EPM environment.

Using a Parameter Interface

When a UIM application is opened, the calling program must specify whether all application programs, called as an exit program or using the CALL dialog command, are passed a single or multiple parameter interface.

When a single parameter is passed, that parameter is a space pointer to a structure containing information for the type of exit or CALL function.

The single parameter interface is used by programs written in languages that can efficiently process data structures. Passing a single parameter is more efficient than passing multiple parameters, and any extensions to the interface are accessible without changing the parameter list of the program being called.

When the single parameter interface is chosen, the structure passed contains a field indicating the level of the structure. Beginning in Version 2 Release 2 (V2R2) of the OS/400 operating system, the structure level is set to two. All parameters and values of a structure are passed when the structure level is equal to or greater than one, unless the parameter description notes otherwise. For more information about choosing the interface level for exit programs, see "Open Display Application (QUIOPNDA) API" on page 28-16 and "Open Print Application (QUIOPNPA) API" on page 28-17.

The interface with multiple parameters is available for programs written in languages that cannot efficiently process data structures. When multiple parameters are passed, each parameter is a space pointer to a piece of information necessary for the type of exit or CALL function.

When the multiple parameter interface is chosen by the program opening the UIM application, the program must also choose which interface level to use. The **interface level** defines the number of parameters passed for a given type of exit or CALL function. This interface allows the parameter interfaces to be extended in the future with upward compatibility for existing application exit programs.

Because the application handle is passed, programs can use dialog variables to access common information. It is more efficient to place one pointer variable that points to a structure or array into the variable pool than to place all

the variables from the structure or array into the variable pool.

Handling Messages

After the UIM calls an application program or runs a control language (CL) command, it handles messages sent to the UIM. Messages sent by the general exit program are moved to an internal UIM program message queue. These messages are referred to as **transient** messages. When the panel is redisplayed by the UIM, the transient messages are shown on the message line of the panel. When the next operation is performed for the panel, in most cases, transient messages are removed from the UIM's internal program message queue and no longer appear in the job log.

Messages sent by any other type of exit program or by a CL command are moved to the program message queue identified on the call to the Display Panel (QUIDSPP) API. These messages are referred to as **user** messages. When the panel is redisplayed by the UIM, the user messages are shown on the message line of the panel. When certain dialog commands are subsequently performed for the panel, the UIM no longer displays the user messages on the panel. It is the responsibility of the application program to remove these messages from the program message queue as appropriate. For more information about when the UIM stops displaying the user messages for a panel, see the *Guide to Programming Displays*.

Escape messages and other accompanying messages are displayed on the panel message line. If the program is sent an escape message to be resent, it needs to do the following:

1. Receive the exception message and save its message reference key.
2. Move all INFO, COMP, and DIAG messages to the previous program message queue.
3. Perform any cleanup processing required for the application program.
4. Resend the exception message using its message reference key.

Whenever an escape message is sent to the UIM from the program or a CL command, the message indicates that the function performed was unsuccessful. The UIM then takes whatever action is appropriate for the unsuccessful function.

CALL Program for a Function Key

Function keys can be assigned to the CALL dialog command on the ACTION attribute of the key item (KEYI) tag. When the function key is pressed, the UIM invokes the specified program.

The CALL dialog command is assigned to a function key to process requests that are specific to the application. These requests either do not have a command interface or require more knowledge about the current application than can normally be passed through a command.

The application can perform most functions, but the application developer must be aware of the effects of changing dialog variables, displaying other panels, and so on.

Single Parameter Interface

Structure level
BINARY(4); positions 1 – 4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5 – 12

Type of call
BINARY(4); positions 13 – 16
Set to the following value:

1 Processes a function key.

Application handle
CHAR(8); positions 17 – 24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25 – 34
The name of the panel currently being processed by the UIM.

Function key pressed
BINARY(4); positions 35 – 38
The function key that is pressed:

1 – 24 A function key (F1 – F24) is processed.
26 The default action when the Enter key is pressed.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Function key pressed	Input	Binary(4)

No additional parameters are required for interface level 2.

CALL Program for a Menu Item

Menu items can be assigned to the CALL dialog command on the ACTION attribute of the menu item (MENUI) tag. When the menu item is selected, the UIM calls the specified program.

The CALL dialog command is assigned to a menu item to process requests that are specific to the application. These requests either do not have a command interface or require more knowledge about the current application than can normally be passed through a command.

The application can perform most functions, but the application developer must be aware of the effects of changing dialog variables, displaying other panels, and so on.

Single Parameter Interface

Structure level
BINARY(4); positions 1 – 4
The interface level supported by this structure, indicating which fields and values are available.

Reserved
CHAR(8); positions 5 – 12

Type of call
BINARY(4); positions 13 – 16
Set to the following value:

2 Processes a menu item.

Application handle
CHAR(8); positions 17 – 24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25 – 34
The name of the panel currently being processed by the UIM.

Menu option
BINARY(4); positions 35 – 38
The option number of the menu item selected by the user.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Menu option	Input	Binary(4)

No additional parameters are required for interface level 2.

CALL Program for an Action List Option and Pull-Down Field Choice

An action list option can specify the CALL dialog command on the ENTER, EXTENTER, PROMPT, and EXTPROMPT attributes of the list action (LISTACT) tag. When processing an action list, the UIM calls the specified program.

A pull-down choice can specify the CALL dialog command on the ACTION attribute of the pull-down field choice (PDFLDC) tag. When processing a selected pull-down choice, the UIM calls the specified program.

The CALL dialog command on an action list or pull-down choice with ACTFOR = LIST on the PDFLDC tag allows the application to handle a request for a single list entry. When processing the list, the UIM calls the program once for each selected entry in the list. Care must be taken in changing other list entries and the current position because the UIM is accessing the same information to perform the remaining action list processing. For example, changing the current position past some selected list entries causes the UIM to not process the skipped ones.

The CALL dialog command on a pull-down choice with ACTFOR = PANEL on the PDFLDC tag allows the application to handle a request related to the panel as a whole. The UIM calls the program only once before redisplaying the panel.

An exception message received by the UIM while performing list processing stops the processing and displays the messages. The action or selection field still contains the option number or selection character and is marked in error.

Single Parameter Interface

Structure level
BINARY(4); positions 1 – 4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5 – 12

Type of call
BINARY(4); positions 13 – 16
Set to one of the following values:

3 A program is called for action list option processing or pull-down choice processing when ACTFOR = LIST is specified on the PDFLDC tag.

9 A pull-down choice. A program is called for pull-down choice processing when ACTFOR = PANEL is specified on the PDFLDC tag.

Application handle
CHAR(8); positions 17 – 24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25 – 34
The name of the panel currently being processed by the UIM.

List name
CHAR(10); positions 35 – 44
The name of the list currently being processed by the UIM. Blanks indicate that the pull-down choice being processed does not operate against a list entry because ACTFOR = PANEL is specified on the PDFLDC tag.

List entry handle
CHAR(4); positions 45 – 48
The list entry handle currently being processed by the UIM. Hexadecimal zeros indicate that the pull-down choice being processed does not operate against a list entry.

The following value may be used:

EXTE Processes the extended action entry.

Option number
BINARY(4); positions 49 – 52
The option number of the list action or pull-down choice being processed.

Function qualifier
BINARY(4); positions 53 – 56
One of the following values:

0 Performs the action specified on the ENTER or EXTENTER attribute of the LISTACT tag or the ACTION attribute of the PDFLDC tag.

–10 Performs the action specified on the PROMPT or EXTPROMPT attribute of the LISTACT tag.

Pull-down field name
CHAR(10); positions 57 – 66
The name of the pull-down field from which the pull-down choice is selected. If this pull-down field does not have a name, this field is set to blanks.

This field is only available when the field for the structure level is set to two or greater.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	List name	Input	Char(10)
5	List entry handle	Input	Char(4)
6	Option number	Input	Binary(4)
7	Function qualifier	Input	Binary(4)

Additional parameter for interface level 2:

8	Pull-down field name	Input	Char(10)

Exit Program for General Panel Checking

A general exit program may be specified on the USREXIT attribute of the panel definition (PANEL) tag. This attribute specifies the name of a dialog variable identifying the program to call.

The UIM calls the program:

- After any necessary validity checking is done by the UIM

  For a VARUPD = YES attribute on the key item (KEYI) or pull-down field choice (PDFLDC) tags, the UIM has already performed validity checking on all displayed values. If errors are found, the UIM does not call the exit program. The current panel is automatically redisplayed with the appropriate message and error highlighting.

  For a VARUPD = NO attribute, the UIM saves all values from entry fields in their displayed form and does not update the dialog variables. The exit program does not have access to these values.

- After the UIM determines what function to perform

  For some dialog commands, the UIM does not call the exit program. For more information on when the UIM calls the exit program, see *Guide to Programming Displays*.

  Any errors detected during function determination cause the UIM to redisplay the panel without calling the exit program.

- Before the UIM performs the determined function, including dialog variable substitution in command strings specified in the CMD dialog command.

The UIM calls the general exit program and passes information about the function. The exit program performs its task and returns control to the UIM through a normal

return or by sending a CPF6A02 or CPF6A03 status message.

If the exit program returns control by sending a CPF6A02 status or escape message, the UIM does not perform the determined function. The panel is redisplayed and any previous messages sent to the UIM are shown.

If the exit program returns control by sending a CPF6A03 status or escape message, the UIM continues performing the determined function. Any previous messages sent to the UIM by the general exit program are shown if and when the function completes.

If the exit program returns control normally, the UIM continues performing the determined function. Any messages sent to the UIM by the exit program are ignored.

Messages sent to the UIM are displayed only once. They are cleared when the next dialog command is performed, excluding the HELP, PAGEUP, PAGEDOWN, and the PRINT dialog commands. If the exit program marks dialog variables in error, the exit program or other application code is responsible for resetting the dialog variables when the variable is no longer incorrect.

The general exit program allows applications to perform extended validity checking on field values. However, other functions are possible because the exit program has access to panel information used by the UIM to perform a specified function. The exit program can prevent the UIM from processing the function. For example, the exit program could intercept a function key, perform some other function, and tell the UIM to not process the function key. While the UIM does not prevent this type of thing, it is the application developer's responsibility to understand enough about UIM processing to ensure that things work properly.

The action performed by the UIM is determined before the exit program is called. Some actions, such as whether or not to perform the action list processing, depend on the contents of dialog variables or list entries. Changes to these variables or list entries do not cause the UIM to evaluate the determined action again.

Dialog variable substitution into command strings is not done until after the exit program is called; changing dialog variables can affect the final, submitted command.

The exit and cancel flags for the job are not reset before the exit program is called, and they are not checked after the exit program returns control to the UIM. Therefore, exit and cancel flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1 – 4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5 – 12

Type of call
BINARY(4); positions 13 – 16
Set to the following value:

4 Processes a general panel exit.

Application handle
CHAR(8); positions 17 – 24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25 – 34
The name of the panel currently being processed by the UIM.

Function key pressed
BINARY(4); positions 35 – 38
The function key that is pressed:

1 – 24 A function key (F1 – F24) is pressed.
26 The Enter key is pressed.
28 The Page Down key is pressed.
29 The Page Up key is pressed.
31 The Home key is pressed.

The general exit program is not called for any other keys.

Function key qualifier
BINARY(4); positions 9 – 42
Provides information about the function that will be performed unless the exit program returns through a CPF6A02 message:

– 1 Submits a command from the command line.

– 2 Performs list action processing. The ACTOR attribute of the list area (LIST) tag determines whether the processing is to be performed by the UIM or by the application program displaying the panel.

– 3 Processes a menu item selection.

– 4 The Enter key is pressed and there is no function for the UIM to perform. The default enter action specified on the ENTER attribute of the display panel (PANEL) tag will be performed.

– 5 Processes a cursor-sensitive prompt function.

– 6 Processes a selection from a pull-down choice. This value is used only when the structure level is two or greater.

– 7 Processes the default selection action specified on the SELECT attribute of the PANEL tag because the Enter key was pressed or the user selected one or more entries in an action list or selection list, and there was no function for the UIM to perform.

Option number
BINARY(4); positions 43 – 46
When the field for the function key qualifier indicates that the UIM is processing a menu item, this field is set to the option number of the menu item selected by the user.

When the field for the function key qualifier indicates that the UIM is processing a pull-down choice, this field is set to the option number of the pull-down choice selected by the user.

For all other values for the function key qualifier, this field is not set.

Pull-down field name
CHAR(10); positions 47 – 57
The name of the pull-down field from which the pull-down choice is selected. If this pull-down field does not have a name, it is set to blanks.

For all other values of the function key qualifier, this field is not set.

This field is only available when the field for the structure level is set to two or greater.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Function key pressed	Input	Binary(4)
5	Function key qualifier	Input	Binary(4)
6	Option number	Input	Binary(4)

Additional parameter for interface level 2:

7	Pull-down field name	Input	Char(10)

Exit Program for an Action List Option or Pull-Down Field Choice

An exit program can be specified for an action list option using the USREXIT attribute of the list action (LISTACT) tag. The exit program is called immediately after the action specified on the ENTER, PROMPT, EXTENTER, or EXTPROMPT attribute of the LISTACT tag is processed.

An exit program for a pull-down field choice can be specified using the USREXIT attribute of the pull-down field choice (PDFLDC) tag. The exit program is called immediately after the action specified on the ACTION attribute of the PDFLDC tag is processed.

Normally, this exit program adds, updates, or removes a list entry for the application when the action list option or pull-down choice action was a command string. The exit program should not modify list entries other than the one currently being processed. If it does, remaining UIM list processing may not perform as expected. If the exit program changes the current entry for the list by adding a new entry to the list, the exit program should reset the current entry to the one currently being processed before returning control to the UIM. If the exit program does not do this, remaining UIM list processing may not perform as expected.

Exception messages received by the UIM from the exit program cause the UIM to stop list processing, and display the messages when the panel is reshown. If the dialog variable identifying the exit program is null (for a pointer variable) or blanks (for a character variable), no exception is reported by the UIM and processing continues as if the USREXIT attribute is not specified.

The cancel and exit flags for the job are not reset before the exit program is called, and the flags are not checked after the exit program returns control to the UIM. Therefore, the cancel and exit flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1 – 4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5 – 12

Type of call
BINARY(4); positions 13 – 16
Set to the following value:

5 The exit program is called for an action list option or pull-down choice when ACTFOR = LIST is specified on the PDFLDC tag.

Application handle
CHAR(8); positions 17 – 24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25 – 34
The name of the panel currently being processed by the UIM.

List name
CHAR(10); positions 35 – 44
The name of the list currently being processed by the UIM.

List entry handle
CHAR(4); positions 45 – 48
The handle of the list entry being processed by the UIM. The following value may be used:

EXTE Processes the extended action entry.

Option number
BINARY(4); positions 49 – 52
The option number of the list action or pull-down choice being processed for the specified list entry.

Function qualifier
BINARY(4); positions 53 – 56
Set to one of the following values:

0 Performs the action specified on the ENTER or EXTENTER attribute of the list action (LISTACT) tag or the ACTION attribute of the PDFLDC tag.

–10 Performs the action specified on the PROMPT or EXTPROMPT attribute of the LISTACT tag.

Action results
BINARY(4); positions 57 – 60
Set to one of the following values, indicating whether or not the list action or pull-down choice was successful:

0 The list action or pull-down choice is successful.
1 The list action or pull-down choice is unsuccessful.

A list action or pull-down choice is considered unsuccessful if any of the following occurs:

- An exception message is sent by the action program or command.
- The cancel flag for the job is set on by the action program or command.
- The exit flag for the job is set on by the action program or command.
- The list action or the pull-down choice sent the CPF6A01 status or escape message to stop list action processing.

Conversely, the list action is successful if none of the above occur.

Pull-down field name
CHAR(10); positions 61 – 70
The pull-down field name from which the pull-down choice is selected. If the pull-down choice does not have a name, this field is set to blanks.

When the action being processed is not a pull-down choice, this field is set to blanks.

This field is available only when the structure level field is set to two or greater.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	List name	Input	Char(10)
5	List entry handle	Input	Char(4)
6	Option number	Input	Binary(4)
7	Function qualifier	Input	Binary(4)
8	Action results	Input	Binary(4)

Additional parameter for interface level 2:

9	Pull-down field name	Input	Char(10)

Exit Program for an Incomplete List

An exit program for handling an incomplete list is defined using the Set List Attributes (QUISETLA) API. A dialog variable is specified identifying the program to call.

The dialog variable cannot be blanks or nulls. The UIM cannot process an incomplete list without the exit program.

This exit allows an application to display part of a list without having to build the entire list. It must either add more entries or mark the list as complete in the direction being extended. If the list is not marked complete and fewer entries than the minimum requested by the UIM are added, the exit program is called again. If the list is not marked complete and no entries are added by the exit program, a CPF6A95 error is reported to the program that called the Display Panel (QUIDSPP), Get List Entry (QUIGETLE), Get List Multiple Entries (QUIGETLM), or Remove List Entry (QUIRMVLE) API.

The exit program can send messages to the UIM for display on the panel. To have the messages shown, the exit program must return control to the UIM by sending a CPF6A05 status or escape message. In this case, any messages sent to the UIM by the incomplete list exit program are shown when the panel is displayed. Otherwise, the exit program returns control to the UIM by doing a normal return.

Sometimes the exit program for an incomplete list is called when no panel is displayed; such a call occurs when the QUIGETLE, the QUIGETLM, or the QUIRMVLE API is used. If no panel is displayed when the exit program sends a CPF6A05 status or escape message, any messages sent to the UIM by the exit program are moved to the program message queue for the program calling that API.

The exit and cancel flags for the job are not reset before the exit program is called, and the flags are not checked after the exit program returns control to the UIM. Therefore, the exit and cancel flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1 – 4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5 – 12

Type of call
BINARY(4); positions 13 – 16
Set to the following value:

6 Processes an incomplete list exit program.

Application handle
CHAR(8); positions 17 – 24
The application handle of the application currently being processed by the UIM.

Reserved
CHAR(10); positions 25 – 34

List name
CHAR(10); positions 35 – 44
The name of the list currently being processed by the UIM.

Incomplete list direction
BINARY(4); positions 45 – 48
The direction from the current list entry in which additional entries must be added.

Set to one of the following values:

0 More list entries are required after the current list entry.
1 More list entries are required before the current list entry.

Number of entries required
BINARY(4); positions 49 – 52
The minimum number of entries that must be added to the list.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	List name	Input	Char(10)
4	Incomplete list direction	Input	Binary(4)
5	Number of entries required	Input	Binary(4)

No additional parameters are required for interface level 2.

Exit Program for Application Formatted Data

The exit program can update the data formatted by the application every time a panel is displayed, and returns control to the UIM through a normal return. The UIM then finishes formatting the panel and displays it. The area formatted by the application is displayed as defined by the contents of the dialog variable.

An application format exit program can be specified on the USREXIT attribute of the application formatted area (APPFMT) tag. This attribute specifies the name of a dialog variable identifying the program to call.

This exit program is called during the formatting of the panel whenever the panel is displayed. If the dialog variable identifying the program is nulls for a pointer variable or blanks for a character variable, no exception is reported and processing continues as if the USREXIT attribute is not specified.

The exit and cancel flags for the job are not reset before the exit program is called, and the flags are not checked after the exit program returns control to the UIM. Therefore, the exit and cancel flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1 – 4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5 – 12

Type of call
BINARY(4); positions 13 – 16
Set to the following value:

7 The exit program is called to update the application formatted data.

Application handle
CHAR(8); positicns 17 – 24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25 – 34
The name of the panel currently being processed by the UIM.

Panel bidirectional orientation
CHAR(1); positions 35 – 35
Identifies the attribute orientation of the panel group.

Set to one of the following values:

N BIDI = NONE is specified or the default on the panel group (PNLGRP) tag.
L BIDI = LTR is specified on the PNLGRP tag.
R BIDI = RTL is specified on the PNLGRP tag.

Device code page
BINARY(4); positions 36 – 39
The number of the code page of the requester's display device.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Panel bidirectional orientation	Input	Char(1)
5	Device code page	Input	Binary(4)

No additional parameters are required for interface level 2.

Exit Program for a Cursor-Sensitive Prompt

An exit program for handling a cursor prompt request can be specified on the PROMPT attribute of the data item (DATAI), data item extender (DATAIX), and list column (LISTCOL) tags. This attribute specifies the name of a dialog variable identifying the program to call.

This dialog variable cannot be blanks or nulls. The UIM cannot process a prompt request without the exit program.

Normally, the exit program displays a selection list panel. Such a list panel allows the user to select one or more entries from a list of possible choices. The application then sets the selected choices in the appropriate dialog variables, which are shown when the UIM redisplays the panel where the prompt function was requested.

The exit and cancel flags for the job are not reset before the exit program is called, and the flags are not checked after the exit program returns control to the UIM. Therefore, the exit and cancel flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1 – 4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5 – 12

Type of call
BINARY(4); positions 13 – 16
Set to the following value:

8 The exit program is called to provide cursor-sensitive prompting.

Application handle
CHAR(8); positions 17 – 24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25 – 34
The name of the panel currently being processed by the UIM.

Panel element type
CHAR(1); positions 35 – 35
One of the following values identify the type of panel element prompt:

1 Prompts for a data item (DATAI tag).
2 Prompts for a data item extender (DATAIX tag).
3 Prompts for a list column (LISTCOL tag).

Reserved
CHAR(1); positions 36 – 36

Dialog variable name
CHAR(10); positions 37 – 46
The name of the dialog variable where the cursor was positioned when the prompt function was requested.

List name
CHAR(10); positions 47 – 56
The name of the list where the cursor was positioned when the prompt function was requested. This field is set only when the panel element type field indicates a LISTCOL tag is prompted.

List entry handle
CHAR(4); positions 57 – 60
The list entry handle where the cursor was positioned when the prompt function was requested. This field is set only when the panel element type field indicates a LISTCOL tag is prompted.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Panel element type	Input	Char(1)
5	Dialog variable name	Input	Char(10)
6	List name	Input	Char(10)

Additional parameter for interface level 2:

7	List entry handle	Input	Char(4)

Part 15. User Object APIs

Chapter 30. User Space APIs

This chapter describes the user space APIs. You can use these APIs to create, change, delete, and retrieve information from user spaces.

User spaces are permanent objects that are located in the user domain. You can save and restore user spaces to other systems. However, if the user spaces contain pointers, you cannot restore the pointers even if you want to restore them to the same system.

The user space APIs include the following:

Change User Space (QUSCHGUS) changes the contents of a user space.
Create User Space (QUSCRTUS) creates a user space in the user domain.
Delete User Space (QUSDLTUS) deletes user spaces created with the QUSCRTUS API.
Retrieve Pointer to User Space (QUSPTRUS) retrieves a pointer to the beginning of a user space for a high-level language (HLL) that supports pointers. HLLs that support pointers can use these pointers to manipulate the contents of a user space directly.
Retrieve User Space (QUSRTVUS) retrieves the contents of a user space. It does not receive descriptive information about the user space, such as its size.

The APIs in this chapter are presented in alphabetical order.

Change User Space (QUSCHGUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Starting position	Input	Binary(4)
3	Length of data	Input	Binary(4)
4	New value	Input	Char(*)
5	Force changes to auxiliary storage	Input	Char(1)

Optional Parameter:

6	Error code	I/O	Char(*)

The Change User Space (QUSCHGUS) API changes the contents of the user space (*USRSPC) object by moving a specified amount of data to the object. For example, you may want to change the contents of a user space for languages that do not support pointers.

Note: To determine the starting position for the QUSCHGUS API, you must add 1 to the offset value. In contrast to the OS/400 list APIs, which use an offset value based on 0 for the starting position, the QUSCHGUS API uses a value based on 1. For the QUSCHGUS API, the first character in the user space is position 1.

Authorities and Locks

Library Authority *USE
User Space Authority *CHANGE
User Space Lock *EXCLRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The special values supported for the library name are *LIBL and *CURLIB.

Starting position
INPUT; BINARY(4)
The first byte of the user space that is to be changed. It must have a value greater than 0.

Length of data
INPUT; BINARY(4)
The length of the new value parameter. The length must be greater than 0.

New value
INPUT: CHAR(*)
The new data to be placed into the user space. The field must be at least as long as the length of data parameter.

Force changes to auxiliary storage
INPUT; CHAR(1)
The method of forcing changes made to the user space to auxiliary storage. The valid values are as follows:

0 Does not force changes. Normal system management writes the changes to auxiliary storage.
1 Forces changes asynchronously. This interrupts the normal system management and ensures that the user space is written to auxiliary storage.
2 Forces changes synchronously. This interrupts the normal system management and ensures that the user space is written immediately to auxiliary storage.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF3C0F E Value &1 for starting position parameter is not valid.
CPF3C04 E User space &1 not changed.
CPD3C0F D Value &1 for starting position parameter is not valid.
CPD3C12 D Number of bytes to change parameter, &1, is not valid.
CPD3C13 D Value &1 for force option is not valid.
CPD3C14 D Starting position &1 and length &2 cause space overflow.
CPD3C15 D New value &1 is shorter than the length specified.
CPD3C17 D Error occurred with source variable.
CPF3C12 E Number of bytes to change parameter, &1, is not valid.
CPF3C13 E Value &1 for force option is not valid.
CPF3C14 E Starting position &1 and length &2 cause space overflow.
CPF3C15 E New value &1 is shorter than the length specified.
CPF3C17 E Error occurred with source variable.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9812 E File &1 in library &2 not found.
CPF9814 E Device &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9821 E Not authorized to program &1 in library &2.
CPF9822 E Not authorized to file &1 in library &2.
CPF9825 E Not authorized to device &1.
CPF9830 E Cannot assign library &1.
CPF9831 E Cannot assign device &1.

Create User Space (QUSCRTUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Extended attribute	Input	Char(10)
3	Initial size	Input	Binary(4)
4	Initial value	Input	Char(1)
5	Public authority	Input	Char(10)
6	Text description	Input	Char(50)

Optional Parameter Group:

7	Replace	Input	Char(10)
8	Error code	I/O	Char(*)

The Create User Space (QUSCRTUS) API creates a user space in the user domain. You can use the QUSCRTUS API to:

- Allow list APIs to generate lists of data.
- Give several jobs access to the same data at the same time.
- Store data and pointers.
- Save and restore the data using CL commands.
- Create a user space as large as 16 megabytes. You cannot create a data area larger than 2000 bytes.
- Pass data from job to job or from system to system.

The user space objects you create are larger than or equal to the size specified. They have a fixed length and can be extended or truncated using the Modify Space (MODS) MI instruction.

Authorities and Locks

User Space Authority
*OBJMGT, *OBJEXIST, and *READ. These authorities are required only if the replace parameter is used and if there is an existing user space to replace.

User Space Library Authority
*USE and *ADD. *OBJOPR plus *READ is equivalent to *USE.

User Space Lock
*EXCL. This applies to both the user space being created and an existing user space being replaced.

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The only special value supported for the library name is *CURLIB.

User spaces created in the QTEMP and QRPLOBJ libraries are not forced to permanent storage; they are deleted when those libraries are cleared at sign-off and system IPL, respectively.

Extended attribute
INPUT; CHAR(10)
The extended attribute of the user space. For example, an object type of *FILE has an extended attribute of PF (physical file), LF (logical file), DSPF (display file), SAVF (save file), and so on.

The extended attribute must be a valid *SNAME. You can enter this parameter in uppercase, lowercase, or mixed case. The API converts it to uppercase.

Initial size
INPUT; BINARY(4)
The initial size of the user space being created. This value must be from 1 byte to 16 776 704 bytes.

Initial value
INPUT; CHAR(1)
The initial value of all bytes in the user space.

Public authority
INPUT; CHAR(10)
The authority you give users who do not have specific private or group authority to the user space. Once the user space has been created, its public authority stays the same when it is moved to another library or restored from backup media.

If the replace parameter is used and a user space exists to be replaced, this parameter is ignored. All authorities are transferred from the replaced user space to the new one.

The valid values for this parameter are:

***ALL**
The user can perform all authorized operations on the object.

Authorization list name
The user space is secured by the specified authorization list, and its public authority is set to *AUTL. The specified authorization list must exist on the system when this API is issued. If it does not exist, the create process fails, and an exception is returned to the application.

***CHANGE**
The user can read the object description and has read, add, update, and delete authority to the object.

***EXCLUDE**
The user cannot access the object in any way.

***LIBCRTAUT**
The public authority for the user space is taken from the CRTAUT value for the target library when the object is created. If the CRTAUT value for the library changes later, that change does not affect user spaces already created. If the CRTAUT value contains an authorization list name and that authorization list secures an object, do not delete the list. If you do, the next time you call this API with the *LIBCRTAUT parameter, it will fail.

***USE**
The user can read the object and its description but cannot change them.

Text description
INPUT; CHAR(50)
This text briefly describes the user space.

Optional Parameter Group

Replace
INPUT; CHAR(10)
If the user space already exists, it is replaced by a new user space of the same name and library, and subject to the same authorities. The replaced user space is renamed and moved to the QRPLOBJ library, which is cleared at system IPL. For details about authorities, ownership, and renaming, see the discussion of the REPLACE parameter in Volume 1 of the *CL Reference*.

Valid values for this parameter are:

***NO** Do not replace an existing user space of the same name and library.
***YES** Replace an existing user space of the same name and library.

If this parameter is left out and the user space to be created already exists, an exception is returned to the application.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF2143 E Cannot allocate object &1 in &2 type *&3.
CPF2144 E Not authorized to &1 in &2 type *&3.
CPF2283 E Authorization list &1 does not exist.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C01 E User space &2 in library &1 not created.
CPD3C01 D Object name &1 is not valid.
CPD3C03 D Extended attribute &1 is not valid.
CPD3C04 D Value &1 for size parameter is not valid.
CPD3C05 D Value &1 for authority parameter is not valid.
CPF3C2B E Extended attribute &1 is not valid.
CPF3C2C E Value &1 for size parameter is not valid.
CPF3C2D E Value &1 for authority parameter is not valid.
CPF3C29 E Object name &1 is not valid.
CPF3C34 E Value &1 for replace option is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.

| CPF9838 E User profile storage limit exceeded.
| CPF9870 E Object &2 type *&5 already exists in library &3.

Delete User Space (QUSDLTUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Error code	I/O	Char(*)

The Delete User Space (QUSDLTUS) API deletes user spaces created with the Create User Space (QUSCRTUS) API. The Delete User Space (DLTUSRSPC) command performs the same function as the QUSDLTUS API.

Authorities and Locks

Library Authority *READ
User Space Authority *OBJEXIST and *USE
User Space Lock *EXCL

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the user space and the name of the library in which it resides. The first 10 characters contain the user space name, and the second 10 characters contain the library name.

The user space name can be either a specific name or a generic name, a string of one or more characters followed by an asterisk (*). If you specify a generic name, QUSDLTUS deletes all user spaces that have names beginning with the string for which the user has authority.

You can use these special values for the library name:

***ALL**	All libraries
***ALLUSR**	All user-defined libraries, plus libraries containing user data and having names starting with Q
***CURLIB**	The job's current library
***LIBL**	The library list
***USRLIBL**	The user portion of the job's library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

| CPF2105 E Object &1 in &2 type *&3 not found.
| CPF2110 E Library &1 not found.
| CPF2113 E Cannot allocate library &1.
| CPF2114 E Cannot allocate object &1 in &2 type *&3.
| CPF2117 E &4 objects type *&3 deleted. &5 objects not deleted.
| CPF2125 E No objects deleted.
| CPF2176 E Library &1 damaged.
| CPF2182 E Not authorized to library &1.
| CPF2189 E Not authorized to object &1 in &2 type *&3.
| CPF3CF1 E Error code parameter not valid.

Retrieve Pointer to User Space (QUSPTRUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Return pointer	Output	PTR(SPP)

Optional Parameter:

3	Error code	I/O	Char(*)

The Retrieve Pointer to User Space (QUSPTRUS) API retrieves a pointer to the contents of a user space so that the data in that space can be directly manipulated by HLL programs that support pointers, such as PL/I, Pascal, C/400, and System C/400 PRPQ.

The QUSPTRUS API returns a pointer even to an object that is subject to an exclusive (*EXCL) lock.

If you create application programs using HLLs that can directly update user spaces using pointers (instead of using the Change User Space (QUSCHGUS) API), you should use your own synchronization data methods using the LOCK or LOCKSL machine interface (MI) instruction or the Allocate Object (ALCOBJ) command to avoid updates at the same time to the same location within a user space. You should also call the Change or Retrieve User Space API to update the object usage information (such as last changed date, last date used, and so on).

Authorities and Locks

Library Authority *USE
User Space Authority *USE

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The special values supported for the library name are *LIBL and *CURLIB.

Return pointer
OUTPUT; PTR(SPP)
The variable containing the pointer to the user space after the QUSPTRUS API has completed running. This parameter must be on a 16-byte boundary alignment.

If an error occurs during processing, this parameter is set to the null pointer.

Optional Parameter

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C05 E One or more errors found while trying to retrieve a pointer.
CPF3C18 E Pointer parameter is not on a 16-byte boundary.
CPD3C18 D Pointer parameter is not on a 16-byte boundary.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.

Example: Retrieving a Pointer with a Pascal Program

To retrieve the pointer to a user space with a call from a Pascal program, specify the following:

```
(*********************************************************************)
(***                                                               ***)
(*** PROGRAM:  POINTER                                            ***)
(***                                                               ***)
(*** LANGUAGE:  PASCAL                                            ***)
(***                                                               ***)
(*** DESCRIPTION:  USE A POINTER TO RETRIEVE CONTENTS OF A USER   ***)
(***              SPACE.                                           ***)
(***                                                               ***)
(*** APIs USED:  QUSPTRUS                                         ***)
(***                                                               ***)
(*********************************************************************)
program Pointer;

  type
    UserSpaceName    = packed array(.1..20.) of char;

    theInformation = record
                       field1: packed array(.1..100.) of char;
                       field2: Integer;
                       field3: Integer;
                     end;
    InfoPtrType    = @theInformation;

  var
    theUserSpace           : UserSpaceName;
    thePtr                 : InfoPtrType;

    ScratchVar             : Integer;

  procedure QUSPTRUS (var UserSpace: UserSpaceName;
                      var PtrToInfo : InfoPtrType);
                      nonpascal;

begin
  theUserSpace := 'TEMPSPACE QGPL';

  QUSPTRUS(theUserSpace,thePtr);

  ScratchVar:=thePtr@.Field2;

end.
```

Additional examples of the API are in Appendix A, "Examples."

Retrieve User Space (QUSRTVUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Starting position	Input	Binary(4)
3	Length of data	Input	Binary(4)
4	Receiver variable	Output	Char(*)

Optional Parameter:

5	Error code	I/O	Char(*)

The Retrieve User Space (QUSRTVUS) API allows you to retrieve the contents of a user space. The QUSRTVUS API does not retrieve descriptive information about the user space object, such as its size. To retrieve information about the attributes of a user space, use the Materialize Space (MATS) machine interface (MI) instruction, the Retrieve Object Description (QUSROBJD) API described on page 33-9, or the Retrieve Object Description (RTVOBJD) command.

If you are repeatedly accessing the contents of a user space and are using an HLL that supports pointers, see "Retrieve Pointer to User Space (QUSPTRUS) API" on page 30-4; this API provides a pointer to the user space for improved performance. When you have obtained a pointer, you use pointer arithmetic to access the contents of a user space.

Note: To determine the starting position for the QUSRTVUS API, you must add 1 to the offset value. In contrast to the OS/400 list APIs, which use an offset value based on 0 for the starting position, the QUSRTVUS API uses a value based on 1. For the QUSRTVUS API, the first character in the user space is position 1.

Authorities and Locks

Library Authority *USE
User Space Authority *USE
User Space Lock *SHRNUP

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The special values supported for the library name are *LIBL and *CURLIB.

Starting position
INPUT; BINARY(4)
The first byte of the user space to be retrieved. A value of 1 will identify the first character in the user space.

Length of data
INPUT; BINARY(4)
The length of the data retrieved. This length must not be larger than the size of the variable that is to receive the data. It must also be greater than 0.

Receiver variable
OUTPUT; CHAR(*)
The variable that will receive the contents of the user space being retrieved.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF3C0F E Value &1 for starting position parameter is not valid.
CPF3C06 E Information not retrieved from user space &1.
CPD3C0F D Value &1 for starting position parameter is not valid.
CPD3C12 D Number of bytes to change parameter, &1, is not valid.
CPD3C14 D Starting position &1 and length &2 cause space overflow.
CPD3C16 D Receiver area too small for length of data &1 specified.
CPD3C20 D Error occurred with receiver variable specified.
CPF3C12 E Number of bytes to change parameter, &1, is not valid.
CPF3C14 E Starting position &1 and length &2 cause space overflow.
CPF3C16 E Receiver area too small for length of data &1 specified.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9812 E File &1 in library &2 not found.
CPF9814 E Device &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9821 E Not authorized to program &1 in library &2.
CPF9822 E Not authorized to file &1 in library &2.
CPF9825 E Not authorized to device &1.
CPF9830 E Cannot assign library &1.
CPF9831 E Cannot assign device &1.

Chapter 31. User Index APIs

This chapter describes the user index APIs, which allow you to create and delete user indexes. A **user index** is an object that allows search functions and automatically sorts data based on the value of the data. User indexes are permanent objects in the user domain. They have an object type of *USRIDX and a maximum size of 4 gigabytes (4 294 967 296 bytes). They help streamline table searching, cross-referencing, and ordering of data. In general, if your table is longer than 1000 entries, an index performs faster than a user-sorted table.

You must provide programs to find and insert index entries. User index entries cannot contain a pointer. You can save and restore all the data in an index. You can also save and restore user indexes to another system.

The user index APIs are:

Create User Index (QUSCRTUI)
Delete User Index (QUSDLTUI)

You can also work with user indexes through the System C/400 PRPQ or machine interface (MI) instructions and the Delete User Index (DLTUSRIDX) command. See the *Languages: System C/400 Programming RPQ P10102 User's Guide and Reference* or the *MI Functional Reference* for details about MI instructions and the *CL Reference* for details about the DLTUSRIDX command.

Create User Index (QUSCRTUI) API

Parameters

Required Parameter Group:

1	User index name and library	Input	Char(20)
2	Extended attribute	Input	Char(10)
3	Entry length attribute	Input	Char(1)
4	Entry length	Input	Binary(4)
5	Key insertion	Input	Char(1)
6	Key length	Input	Binary(4)
7	Immediate update	Input	Char(1)
8	Optimization	Input	Char(1)
9	Public authority	Input	Char(10)
10	Text description	Input	Char(50)

Optional Parameter Group:

11	Replace	Input	Char(10)
12	Error code	I/O	Char(*)

The Create User Index (QUSCRTUI) API creates a user index in the user domain. The user index can have a maximum size of 4 gigabytes. A user index with a size greater than 1 gigabyte cannot be saved to or restored from a release prior to Version 2 Release 2 Modification 0.

If the user index was created prior to Version 2 Release 2 Modification 0, the size of the user index is limited to a maximum of 1 gigabyte. If the user index was created on or after Version 2 Release 2 Modification 0, the size of the object is limited to a maximum of 4 gigabytes. The size is dependent on the amount of storage needed for the number and size of index entries and excludes the size of the associated space, if any.

Note: You can tell whether a user index object can be saved to a release prior to Version 2 Release 2 Modification 0:

- By ensuring the current object size is less than 1 gigabyte by using the Display Object Description (DSPOBJD) API, the List Objects (QUSLOBJ) API, or the Retrieve Object Description (QUSROBJD) API
- By ensuring that the key length field is 120 bytes or less by using the materialize queue attributes (MATQAT) MI instruction.

You can use user indexes to:

- Provide search functions
- Do faster insert operations than in a database file
- Do faster retrieve operations than in a database file
- Create an index by name, such as a telephone directory
- Use order entry programs
- Look up abbreviations in an index
- Sort data automatically

For more information about index considerations, refer to "User Index Considerations" on page 2-11.

Authorities and Locks

Library Authority
*OBJOPR, *ADD, and *READ.

User Index Authority
*OBJMGT, *OBJEXIST, and *READ. These authorities are required only if the replace parameter is used and there is an existing user index to replace.

User Index Lock
*EXCL. This applies to both the user index being created and an existing user index being replaced.

Required Parameter Group

User index name and library
INPUT; CHAR(20)
The name of the user index being created, and the library in which it is to be located. The first 10 characters contain the user index name, and the second 10 characters contain the library name. You can use this special value for the library name:

***CURLIB** The job's current library

User indexes created in the QTEMP and QRPLOBJ libraries are not forced to permanent storage; they are deleted when those libraries are cleared at sign-off and system IPL, respectively.

Extended attribute
INPUT; CHAR(10)
The extended attribute of the user index. For example, an object type of *FILE could have an extended attribute of PF (physical file), LF (logical file), DSPF (display file), or SAVF (save file).

The extended attribute must be a valid *SNAME. You can enter this parameter in uppercase, lowercase, or mixed case. The API automatically converts it to uppercase.

Entry length attribute
INPUT; CHAR(1)
Whether there are fixed-length or variable-length entries in the user index. The valid values are:

F Fixed-length entries
V Variable-length entries

Entry length
INPUT; BINARY(4)
The length for fixed-length entries of each index entry.

This parameter must be 0 or -1 for variable-length entries. Variable-length entries that specify 0 enable a key length between 1 through 120. Variable-length entries that specify -1 enable a key length between 1 through 2000.

The valid values for fixed-length entries are from 1 through 2000.

Note: A user index created with an entry length greater than 120 cannot be saved or restored to a release prior to Version 2 Release 2 Modification 0.

Key insertion
INPUT; CHAR(1)
Whether or not the inserts to the index are by key. The valid values are:

0 No insertion by key
1 Insertion by key

Key length
INPUT; BINARY(4)
The length of the key where the first byte of an entry is the beginning of the key for the index entries. The value for this parameter must be 0 for no insertion by key. If you specify key length insertion, this value is from 1 through 2000.

Immediate update
INPUT; CHAR(1)
Whether or not the updates to the index are written synchronously to auxiliary storage on each update to the index. The valid values are:

0 No immediate update
1 Immediate update

Each update to the index is written to auxiliary storage after every insert and remove operation.

Optimization
INPUT; CHAR(1)
The type of access in which to optimize the index. The valid values are:

0 Optimize for random references
1 Optimize for sequential references

Public authority
INPUT; CHAR(10)
The authority you give to users who do not have specific private or group authority to the user index. Once the user index has been created, its public authority stays the same when it is moved to another library or restored from backup media.

If the replace parameter is used and a user index exists to be replaced, this parameter is ignored. All authorities are transferred from the replaced user index to the new one.

The valid values for this parameter are:

***ALL** The user can perform all authorized operations on the user index.
Authorization list name
The user index is secured by the specified authorization list, and its public authority is set to *AUTL. The specified authorization list must exist on the system when this API is issued. If it does not exist, the create process fails, and an exception is returned to the application.
***CHANGE**
The user has read, add, update, and delete authority for the user index and can read the object description.
***EXCLUDE**
The user cannot access the user index in any way.
***LIBCRTAUT**
The public authority for the user index is taken from the CRTAUT value for the target library when the object is created. If the CRTAUT value for the library changes later, that change does not affect user indexes already created. If the CRTAUT value contains an authorization list name and that authorization list secures an object, do not delete the list. If you do, the next time you call this API with the *LIBCRTAUT parameter, it will fail.
***USE** The user can read the object description and contents but cannot change the user index.

Text description
INPUT; CHAR(50)
A brief description of the user index.

Optional Parameter Group

Replace
INPUT; CHAR(10)
If the user index already exists, it is replaced by a new user index of the same name and library, and subject to the same authorities. The replaced user index is renamed and moved to the QRPLOBJ library, which is cleared at system IPL. For details about authorities,

ownership, and renaming, see the discussion of the REPLACE parameter in Volume 1 of the *CL Reference*.

Valid values for this parameter are:

***NO** Do not replace an existing user index of the same name and library.

***YES** Replace an existing user index of the same name and library.

If this parameter is left out and the user index to be created already exists, an exception is returned to the application.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Some of the parameters are interdependent and are shown in the following table:

Entry Length Attributes	Entry Length (n)	Key Insertion	Key Length (x)
Fixed	1 ≤ n ≤ 2000	No	0
Fixed	1 ≤ n ≤ 2000	Yes	1 ≤ x ≤ n
Variable	0, -1	No	0
Variable	0	Yes	1 ≤ x ≤ 120
Variable	-1	Yes	1 ≤ x ≤ 2000

Error Messages

CPF2143 E Cannot allocate object &1 in &2 type *&3.
CPF2144 E Not authorized to &1 in &2 type *&3.
CPF2283 E Authorization list &1 does not exist.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C0A E Value &1 for entry length parameter is not valid.
CPF3C0B E Value &1 for immediate update parameter is not valid.
CPF3C0D E Value &1 for key insertion parameter is not valid.
CPF3C0E E Value &1 for the optimization parameter is not valid.
CPF3C03 E User index &2 not created.
CPD3C01 D Object name &1 is not valid.
CPD3C02 D Value &1 for entry length attribute parameter is not valid.
CPD3C03 D Extend attribute &1 is not valid.
CPD3C05 D Value &1 for authority parameter is not valid.
CPD3C0A D Value &1 for entry length parameter is not valid.
CPD3C0B D Value &1 for immediate update parameter is not valid.
CPD3C0C D Value &1 for key length parameter is not valid.
CPD3C0D D Value &1 for key insertion parameter is not valid.
CPD3C0E D Value &1 for the optimization parameter is not valid.
CPF3C2A E Value &1 for entry length attribute parameter is not valid.
CPF3C2B E Extended attribute &1 is not valid.
CPF3C2D E Value &1 for authority parameter is not valid.
CPF3C29 E Object name &1 is not valid.
CPF3C34 E Value &1 for replace option is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.
CPF9870 E Object &2 type *&5 already exists in library &3.

Delete User Index (QUSDLTUI) API

Parameters

Required Parameter Group:

1	Qualified user index name	Input	Char(20)
2	Error code	I/O	Char(*)

The Delete User Index (QUSDLTUI) API deletes user indexes created with the Create User Index (QUSCRTUI) API. The Delete User Index (DLTUSRIDX) command has the same function as the QUSDLTUI API.

Authorities and Locks

Library Authority *READ
User Index Authority *OBJEXIST and *USE
User Index Lock *EXCL

Required Parameter Group

Qualified user index name

INPUT; CHAR(20)

The name of the user index and the name of the library in which it resides. The first 10 characters contain the user index name, and the second 10 characters contain the library name. The user index name can be either a specific name or a generic name, a string of one or more characters followed by an asterisk (*). If you specify a generic name, QUSDLTUI deletes all user indexes that have names beginning with the string for which the user has authority.

You can use these special values for the library name:

***ALL** All libraries

***ALLUSR**

All user-defined libraries, plus libraries containing user data and having names starting with Q

***CURLIB**
The job's current library
***LIBL** The library list
***USRLIBL**
The user portion of the job's library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF2105 E Object &1 in &2 type *&3 not found.
CPF2110 E Library &1 not found.
CPF2113 E Cannot allocate library &1.
CPF2114 E Cannot allocate object &1 in &2 type *&3.
CPF2117 E &4 objects type *&3 deleted. &5 objects not deleted.
CPF2125 E No objects deleted.
CPF2176 E Library &1 damaged.
CPF2182 E Not authorized to library &1.
CPF2189 E Not authorized to object &1 in &2 type *&3.
CPF3CF1 E Error code parameter not valid.

Chapter 32. User Queue APIs

This chapter describes the user queue APIs, which let you create and delete user queues. User queues are permanent objects in the user domain with an object type of *USRQ. They provide a way for one or more processes to communicate asynchronously.

You can save and restore user queues. However, you can only save or restore its definition. You cannot save or restore the messages in it. You cannot restore a user queue if a user queue with the same name already exists in the library. You must provide programs to use this object type to enqueue and dequeue messages.

The user queue APIs are:

Create User Queue	(QUSCRTUQ)
Delete User Queue	(QUSDLTUQ)

In addition to using these APIs, you can work with user queues through the System C/400 PRPQ, machine interface (MI) instructions, and the Delete User Queue (DLTUSRQ) command. For details about MI instructions, see the *Languages: System C/400 Programming RPQ P10102 User's Guide and Reference* or the *MI Functional Reference*. For details about the DLTUSRQ command, see the *CL Reference*.

Create User Queue (QUSCRTUQ) API

Parameters

Required Parameter Group:

1	Qualified user queue name	Input	Char(20)
2	Extended attribute	Input	Char(10)
3	Queue type	Input	Char(1)
4	Key length	Input	Binary(4)
5	Maximum message size	Input	Binary(4)
6	Initial number of messages	Input	Binary(4)
7	Additional number of messages	Input	Binary(4)
8	Public authority	Input	Char(10)
9	Text description	Input	Char(50)

Optional Parameter Group:

10	Replace	Input	Char(10)
11	Error code	I/O	Char(*)

The Create User Queue (QUSCRTUQ) API creates a user queue in the user domain, based on the input parameters.

You can use user queues to:

- Communicate between two processes asynchronously
- Store data in arrival sequence for later use
- Contain keyed messages
- Use the batch compile machine
- Control busy resources
- Permit better performance than the data queue interface
- Process a queue by key rather than by FIFO (first-in first-out) or LIFO (last-in first-out)

When user queues are created, they:

- Can grow to a maximum of 16 megabytes in size, though some of that is used for the queue definition.
- Are placed in the same auxiliary storage pool (ASP) as the library.

The system can automatically extend user queues, so you should create a small queue and allow the system to extend the queue as needed. Smaller queues provide better performance and use less storage. However, the system does not automatically reduce the size of the queue if it is too large. If you specify the additional number of messages as 0, the system does not extend the user queue you create when the queue is full.

To decrease the size of a user queue, a user must delete the queue and create it again.

Authorities and Locks

Library Authority
*OBJOPR, *ADD, and *READ.

User Queue Authority
*OBJMGT, *OBJEXIST, and *READ. These authorities are required only if the replace parameter is used and if there is an existing user queue to replace.

User Queue Lock
*EXCL. This applies to both the user queue being created and an existing user queue being replaced.

Required Parameter Group

Qualified user queue name
INPUT; CHAR(20)
The first 10 characters contain the user queue name, and the second 10 characters contain the name of the library where the user queue is located. The only special value supported is *CURLIB.

User queues created in the QTEMP and QRPLOBJ libraries are not forced to permanent storage; they are deleted when those libraries are cleared at sign-off and system IPL, respectively.

Extended attribute
INPUT; CHAR(10)
The extended attribute of the user queue. For example, an object type of *FILE has an extended attribute of PF (physical file), LF (logical file), DSPF (display file), SAVF (save file), and so on.

The extended attribute must be a valid *SNAME. You can enter this parameter in uppercase, lowercase, or mixed case. The API automatically converts it to uppercase.

Queue type
INPUT; CHAR(1)
The sequence in which messages are to be dequeued from the queue. The valid values are:

F First-in first-out
K Keyed
L Last-in first-out

Key length
INPUT; BINARY(4)
The length in bytes of the message key from 1 to 256 if you specify the queue type as keyed. If you specify that the queue is not a keyed queue, the value must be 0.

Maximum message size
INPUT; BINARY(4)
The maximum allowed size of messages to be placed on the message queue. The API truncates messages that are longer than the value you specify. The maximum size allowed is 64 000 bytes.

Initial number of messages
INPUT; BINARY(4)
The initial number of messages that the queue can contain.

Additional number of messages
INPUT; BINARY(4)
The amount to increase the maximum number of messages value when the queue is full. When this parameter is set to 0, the queue is not extended if an overflow occurs and an error message is sent to the program attempting to place a message on the queue.

Public authority
INPUT; CHAR(10)
The authority you give to the users who do not have specific private or group authority to the user queue. Once the user queue has been created, its public authority stays the same when it is moved to another library or restored from backup media.

If the replace parameter is used and a user queue exists to be replaced, this parameter is ignored. All authorities are transferred from the replaced user queue to the new one.

The valid values for this parameter are:

***ALL** The user can perform all authorized operations on the user queue.
Authorization list name
The user queue is secured by the specified authorization list, and its public authority is set to *AUTL. The specified authorization list must exist on the system when this API is issued. If it does not exist, the create process fails, and an exception is returned to the application.
***CHANGE**
The user has read, add, update, and delete authority to the user queue and can read the object description.
***EXCLUDE**
The user cannot access the user queue in any way.
***LIBCRTAUT**
The public authority for the user queue is taken from the CRTAUT value for the target library when the object is created. If the CRTAUT value for the library changes later, that change does not affect user queues already created. If the CRTAUT value contains an authorization list name and that authorization list secures an object, do not delete the list. If you do, the next time you call this API with the *LIBCRTAUT parameter, it will fail.
***USE** The user can read the object description and the user queue's contents but cannot change them.

Text description
INPUT; CHAR(50)
A brief description of the user queue.

Optional Parameter Group

Replace
INPUT; CHAR(10)
If the user queue already exists, it is replaced by a new user queue of the same name and library, and subject to the same authorities. The replaced user queue is renamed and moved to the QRPLOBJ library, which is cleared at system IPL. For details about authorities, ownership, and renaming, see the discussion of the REPLACE parameter in Volume 1 of the *CL Reference*.

Valid values for this parameter are:

***NO** Do not replace an existing user queue of the same name and library.
***YES** Replace an existing user queue of the same name and library.

If this parameter is left out and the user queue to be created already exists, an exception is returned to the application.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF2143 E Cannot allocate object &1 in &2 type *&3.
CPF2144 E Not authorized to &1 in &2 type *&3.
CPF2283 E Authorization list &1 does not exist.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C02 E User queue &1 not created.
CPD3C01 D Object name &1 is not valid.

| | CPD3C03 D Extended attribute &1 is not valid.
| | CPD3C05 D Value &1 for authority parameter is not valid.
| | CPD3C06 D Number of messages requested, &1, is too large for this queue.
| | CPD3C07 D Value &1 for queue type parameter not valid.
| | CPD3C08 D Initial number of queue messages specified, &1, is not valid.
| | CPD3C09 D Extension parameter value &1 for queue overflow is not valid.
| | CPD3C10 D Value &1 for key length parameter is not valid.
| | CPD3C11 D Value &1 for maximum message size parameter is not valid.
| CPF3C08 E Initial number of queue messages specified, &1, is not valid.
| CPF3C09 E Extension parameter value &1 for queue overflow is not valid.
| CPF3C10 E Value &1 for key length parameter is not valid.
| CPF3C11 E Value &1 for maximum message size parameter is not valid.
| CPF3C2B E Extended attribute &1 is not valid.
| CPF3C2D E Value &1 for authority parameter is not valid.
| CPF3C2E E Number of messages requested, &1, is too large for this queue.
| CPF3C2F E Value &1 for queue type parameter not valid.
| CPF3C29 E Object name &1 is not valid.
| CPF3C34 E Value &1 for replace option is not valid.
| CPF3C36 E Number of parameters, &1, entered for this API was not valid.
| CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9838 E User profile storage limit exceeded.
| CPF9870 E Object &2 type *&5 already exists in library &3.

Delete User Queue (QUSDLTUQ) API

Parameters

Required Parameter Group:

1	Qualified user queue name	Input	Char(20)
2	Error code	I/O	Char(*)

The Delete User Queue (QUSDLTUQ) API deletes user queues created with the Create User Queue (QUSCRTUQ) API. The Delete User Queue (DLTUSRQ) command has the same function.

Authorities and Locks

Library Authority *READ
User Queue Authority *OBJEXIST and *USE
User Queue Lock *EXCL

Required Parameter Group

Qualified user queue name
INPUT; CHAR(20)
The name of the user queue and the name of the library in which it resides. The first 10 characters contain the user queue name, and the second 10 characters contain the library name.

The user queue name can be either a specific name or a generic name, a string of one or more characters followed by an asterisk (*). If you specify a generic name, QUSDLTUS deletes all user queues that have names beginning with the string for which the user has authority.

You can use these special values for the library name:

***ALL** All libraries
***ALLUSR** All user-defined libraries, plus libraries containing user data and having names starting with Q
***CURLIB** The job's current library
***LIBL** The library list
***USRLIBL** The user portion of the job's library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

| CPF2105 E Object &1 in &2 type *&3 not found.
| CPF2110 E Library &1 not found.
| CPF2113 E Cannot allocate library &1.
| CPF2114 E Cannot allocate object &1 in &2 type *&3.
| CPF2117 E &4 objects type *&3 deleted. &5 objects not deleted.
| CPF2125 E No objects deleted.
| CPF2176 E Library &1 damaged.
| CPF2182 E Not authorized to library &1.
| CPF2189 E Not authorized to object &1 in &2 type *&3.
| CPF3CF1 E Error code parameter not valid.

Chapter 33. Object APIs

You can use object APIs to obtain information about OS/400 objects. The object APIs and their functions include the following:

Change Object Description (QLICOBJD) changes object information for a specific object.
Convert Type (QLICVTTP) converts an object type to and from hexadecimal format.
List Objects (QUSLOBJ) generates a list of object names and descriptive information based on the specified parameters.
Retrieve Object Description (QUSROBJD) retrieves object information for a specific object.

The QUSLOBJ and QUSROBJD APIs return much of the same information. However, the APIs differ in several respects:

differently among their output formats. For example, the OBJL0300 format of the QUSLOBJ API does not contain exactly the same data as the OBJD0300 format of the QUSROBJD API.

- The APIs use different data formats for some specific items, such as dates and times.
- The APIs differ in efficiency, depending on your application. In most cases, the QUSROBJD API is faster at retrieving information about a single object. The QUSLOBJ API is faster at retrieving information about several objects.

The APIs in this chapter are presented in alphabetical order.

Change Object Description (QLICOBJD) API

Parameters

Required Parameter Group:

1	Returned library name	Output	Char(10)
2	Object and library name	Input	Char(20)
3	Object type	Input	Char(10)
4	Changed object information	Input	Char(*)
5	Error code	I/O	Char(*)

The Change Object Description (QLICOBJD) API lets you change object information for a specific object.

Before the object can be changed with the API, the allow change by program field is checked. If the API cannot be used to change the object, message CPF219B is issued.

When an object has been successfully updated with the API, the date and time the object was changed and the changed by program field are updated.

Authorities and Locks

Library Authority *READ
Non-*FILE Object Authority *OBJMGT
***FILE Object Authority** *OBJOPR and *OBJMGT
Object Lock *EXCLRD

Required Parameter Group

Returned library name
OUTPUT; CHAR(10)
The name of the library that contains the changed object. If *CURLIB, *LIBL, or a name is specified for the library name in the object and library name parameter, the value returned is the name of the library where the object was found. If an error occurred while the API was accessing the object, blanks are returned.

Object and library name
INPUT; CHAR(20)
The object for which you want to change information and the library in which it is located. The first 10 characters contain the object name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The job's library list

Object type
INPUT; CHAR(10)
The type of object for which you want to change the information. You can only specify specific external object types. An asterisk (*) must precede the object type. For a complete list of the available object types, see the *CL Reference*.

Changed object information
INPUT; CHAR(*)
The information for the object that you want to change. The information must be in the following format:

Number of variable length records
BINARY(4)
Total number of all of the variable length records.
Variable length records
The fields of the object's description to change and the data used for the change. For the specific format of the variable length record, see "Format for Variable Length Record."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format for Variable Length Record

The following table defines the format for the variable length records.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Key
4	4	BINARY(4)	Length of data
8	8	CHAR(*)	Data

If the length of the data is longer than the key field's data length, the data will be truncated at the right. No message will be issued.

If the length of the data is smaller than the key field's data length, the data will be padded with blanks at the right. No message will be issued.

Field Descriptions

Data. The data used to change a specific field of the object description. The values specified for the allow-change-by-program and days-used-count-reset keys are validity checked to verify that they are 0 or 1. The data specified for other keys is not validity checked.

Key. Identifies a field of the object's description to change. Only specific fields of the object's description can be changed. See "Keys" for the list of valid keys.

Length of data. The length of the data used to change a specific field of the object's description.

Keys: The following table lists the valid keys for the key field area of the variable length record.

Key	Type	Field
1	CHAR(30)	Source file
2	CHAR(13)	Source file last changed date and time
3	CHAR(13)	Compiler
4	CHAR(8)	Object control level
5	CHAR(13)	Licensed program
6	CHAR(7)	Program temporary fix (PTF)
7	CHAR(6)	Authorized program analysis report (APAR)
8	CHAR(1)	Allow change by program
9	CHAR(10)	User-defined attribute
10	CHAR(50)	Text
11	CHAR(1)	Days used count reset
12	CHAR(4)	Product option load ID
13	CHAR(4)	Product option ID
14	CHAR(4)	Component ID

Field Descriptions

Allow change by program. This field is used to prevent users from changing an object's description with this API. It must have a value of 0 or 1.

0 Changes are not allowed with the API. Once this field has been changed to 0, the API cannot be used to make any further changes to the object's description.
1 Changes are allowed with the API. The API can be used to change the object's description.

Authorized program analysis report (APAR). The authorized program analysis report identification that caused this object to be patched. IBM APARs have an uppercase alphabetic character followed by 5 decimal numbers. If you want to conform with the system, this format should be followed.

Compiler. The name, version level, release level, and modification level of the compiler. Objects created with IBM products will have the licensed program name of the compiler in the compiler name field and a version field in the VxRxMy format where x must be 0-9 and y must be 0-9 or A-Z. If you want to conform with the system, this format should be followed.

Compiler name CHAR(7)
Version CHAR(6)

Component ID. The product administrator owns this field. It can be used to track information about objects, such as object size, at a lower level than the product option ID.

Days used count reset. This field is used to reset the days used count. It must have a value of 0 or 1.

0 The days used count is not reset for the object.
1 The days used count is reset for the object.

Licensed program. The name, version level, release level, and modification level of the licensed program. Objects that are a part of an IBM licensed program have a valid licensed program name (7 characters containing 0-9 and uppercase A-Z). The version field is in the VxRxMy format where x must be 0-9 and y must be 0-9 or A-Z. If you want to conform with the system, this format should be followed.

Licensed program name CHAR(7)
Version CHAR(6)

Object control level. The object control level for the object. IBM programs will have an 8-character decimal value.

Product option ID. Identifies part of a licensed program (product). Products can have multiple options. Objects that are a part of an IBM licensed program must be 0 (*BASE) through 99. If you want to conform with the system, this format should be followed.

Product option load ID. The language identifier associated with the object. Objects that are a part of an IBM licensed program must have one of the allowed languages in the 29xx format. If you want to conform with the system, this format should be followed.

Program temporary fix. The program temporary fix (PTF) that resulted in the creation of the object. For IBM objects the first 2 characters are a prefix ID, and the remaining 5 characters are the program change ID (decimal). The field is blank if the object was not changed because of a PTF. If

| you want to conform with the system, this format should be followed.

| **Source file**. The source file used to create the object, the library name in which it is located, and the name of the source file member.

| *Source file name* CHAR(10)
| *Library name* CHAR(10)
| *Member name* CHAR(10)

| Objects created with IBM products have valid object names for the qualified source file name. If you want to conform with the system, this format should be followed.

| **Source file last changed date and time**. The date and time the member in the source file was last updated. Objects created with IBM products will be in the CYYMMDDHHMMSS format, where:

| *C* Century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
| *YY* Year
| *MM* Month
| *DD* Day
| *HH* Hour
| *MM* Minute
| *SS* Second

| If you want to conform with the system, this format should be followed.

| **Text**. The user-defined text that briefly describes the object and its function.

| **User-defined attribute**. This is an attribute you define. This should not be confused with the extended attribute of the object. The extended attribute is set by the system when an object is created.

| Error Messages

| CPF2101 E Object type *&1 not valid.
| CPF2150 E Object information function failed.
| CPF2151 E Operation failed for &2 in &1 type *&3.
| CPF219B E Cannot change &1 in &2 type *&3.
| CPF219E E Object type *&1 not valid external object type.
| CPF2199 E &2 not valid for field &1.
| CPF24B4 E Severe error while addressing parameter list.
| CPF2451 E Message queue &1 is allocated to another job.
| CPF3CF1 E Error code parameter not valid.
| CPF7304 E File &1 in &2 not changed.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9811 E Program &1 in library &2 not found.
| CPF9812 E File &1 in library &2 not found.
| CPF9814 E Device &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9821 E Not authorized to program &1 in library &2.
| CPF9822 E Not authorized to file &1 in library &2.
| CPF9825 E Not authorized to device &1.
| CPF9830 E Cannot assign library &1.
| CPF9831 E Cannot assign device &1.

| Convert Type (QLICVTTP) API

Parameters

Required Parameter Group:

1	Conversion	Input	Char(10)
2	Symbolic object type	I/O	Char(10)
3	Hexadecimal object type	I/O	Char(2)
4	Error code	I/O	Char(*)

| The Convert Type (QLICVTTP) API lets you convert an object type from the external symbolic format to the internal hexadecimal format and vice versa.

| You can use the QLICVTTP API to:

- | Convert a specific symbolic object type to an equivalent hexadecimal object type
- | Convert a specific hexadecimal object type to an equivalent symbolic object type

| Required Parameter Group

| Conversion
| INPUT; CHAR(10)
| The type of conversion to perform.

| ***HEXTOSYM** An object type in hexadecimal form is converted to an equivalent symbolic object type.
| ***SYMTOHEX** A symbolic object type is converted to an equivalent hexadecimal form.

| Symbolic object type
| I/O; CHAR(10)
| The external symbolic name given to an object type. An asterisk (*) precedes the object type. The value of the conversion parameter specified determines if this is an input or output field. For a complete list of the available object types, see the *CL Reference*.

| Hexadecimal object type
| I/O; CHAR(2)
| The MI representation of an external object type. This field is in hexadecimal form. The value of the conversion specified determines if this is an input or output field. For a list of the available object types in hexadecimal format, see the section headed "OS/400 Object Types and Subtypes" in the *Diagnostic Aids – Volume 1*.

| Error code
| I/O; CHAR(*)
| The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

| Error Messages

| CPF2101 E Object type *&1 not valid.
| CPF2102 E Object type and subtype code &1 not valid.
| CPF219C E Conversion value &1 not valid.
| CPF219D E Object type &1 not valid external object type.
| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.

List Objects (QUSLOBJ) API

Parameters

Required Parameter Group:

1	Qualified user space object	Input	Char(20)
2	Format name	Input	Char(8)
3	Object and library name	Input	Char(20)
4	Object type	Input	Char(10)

Optional Parameter:

5	Error code	I/O	Char(*)

The List Objects (QUSLOBJ) API lets you generate a list of object names and descriptive information based on specified selection parameters. The QUSLOBJ API places the list in the specified user space. The generated list replaces any existing list in the user space.

You can use the QUSLOBJ API to:

- List objects in a library
- List objects of only one type
- Write an application program to move programs from the QRPLOBJ library back to where they were originally located
- Provide backup analysis based on when the object was last saved or last updated
- Provide source member and object analysis from source member information to verify that the current source was used to create the specified object

The QUSLOBJ API returns information in several formats. All formats except OBJL0100 include an information status field that describes the completeness and validity of the information. Be sure to check the information status field before using any other information returned.

Authorities and Locks

User Space Authority *CHANGE
User Space Library Authority
*USE
User Space Lock *EXCLRD
Object Authority Object ownership or any object authority
Object Library Authority
*READ

Required Parameter Group

Qualified user space object
INPUT; CHAR(20)
The name of the *USRSPC object that is to receive the generated list. The first 10 characters contain the user space object name, and the second 10 characters contain the name of the library where the user space is located. The special values supported for the library name are *LIBL and *CURLIB.

Format name
INPUT; CHAR(8)
The format of the information returned on each object that is requested. You must use one of the following format names:

OBJL0100 Object names (fastest)
OBJL0200 Text description and extended attribute
OBJL0300 Basic object information
OBJL0400 Creation information
OBJL0500 Save and restore information
OBJL0600 Usage information
OBJL0700 All object information (slowest)

For details about the formats, see "Format of the Generated Lists" on page 33-5. For performance reasons, you should choose the format that returns only as much information as you need. The higher the number of the format name, the more information is returned and the more time it takes to process.

Object and library name
INPUT; CHAR(20)
The object and library names to place in the *USRSPC object. The first 10 characters contain the object name, which may be a simple name, a generic name, or the special value *ALL.

The second 10 characters identify the name of the library or libraries to search for the specified objects. The following special values are allowed:

***ALL** All libraries
***ALLUSR**
All user-defined libraries, plus libraries containing user data and having names starting with Q
***CURLIB**
The job's current library
***LIBL** The library list
***USRLIBL**
The user portion of the job's library list

Object type
INPUT; CHAR(10)
The types of objects to search for. You may either enter a specific object type, or a special value of *ALL. For a complete list of the available object types, see the *CL Reference*.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code

Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated Lists

The object list consists of:

- A user area
- A generic header
- An input parameter section
- A list data section

For details about the user area and generic header, see "User Space Format for List APIs" on page 2-7. For details about the other items, see the following sections. For a detailed description of each field in the information returned, see "Field Descriptions" on page 33-6.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	Object name specified
38	26	CHAR(10)	Object library name specified
48	30	CHAR(10)	Object type specified

OBJL0100 List Data Section: The following information is returned in the list data section of the OBJL0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 33-6.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name used
10	A	CHAR(10)	Object library name used
20	14	CHAR(10)	Object type used

OBJL0200 List Data Section: The following information is returned in the list data section of the OBJL0200 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 33-6.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0100 format
30	1E	CHAR(1)	Information status
31	1F	CHAR(10)	Extended object attribute
41	29	CHAR(50)	Text description
91	5B	CHAR(10)	User-defined attribute
101	65	CHAR(7)	Reserved

OBJL0300 List Data Section: The following information is returned in the list data section of the OBJL0300 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 33-6.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0200 format
108	6C	BINARY(4)	Auxiliary storage pool
112	70	CHAR(10)	Object owner
122	7A	CHAR(2)	Object domain
124	7C	CHAR(8)	Creation date and time
132	84	CHAR(8)	Change date and time
140	8C	CHAR(10)	Storage
150	96	CHAR(1)	Object compression status
151	97	CHAR(1)	Allow change by program
152	98	CHAR(1)	Changed by program
153	99	CHAR(19)	Reserved

OBJL0400 List Data Section: The following information is returned in the list data section of the OBJL0400 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 33-6.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0300 format
172	AC	CHAR(10)	Source file name
182	B6	CHAR(10)	Source file library name
192	C0	CHAR(10)	Source file member name
202	CA	CHAR(13)	Source file updated date and time
215	D7	CHAR(10)	Creator's user profile
225	E1	CHAR(8)	System where object was created
233	E9	CHAR(9)	System level
242	F2	CHAR(16)	Compiler
258	102	CHAR(8)	Object level
266	10A	CHAR(1)	User changed
267	10B	CHAR(16)	Licensed program
283	11B	CHAR(10)	Program temporary fix (PTF)
293	125	CHAR(10)	Authorized program analysis report (APAR)
303	12F	CHAR(21)	Reserved

OBJL0500 List Data Section: The following information is returned in the list data section of the OBJL0500 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 33-6.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0400 format
324	144	CHAR(8)	Object saved date and time
332	14C	CHAR(8)	Object restored date and time
340	154	BINARY(4)	Save size
344	158	BINARY(4)	Save size multiplier
348	15C	BINARY(4)	Save sequence number
352	160	CHAR(10)	Save command
362	16A	CHAR(71)	Save volume ID
433	1B1	CHAR(10)	Save device
443	1BB	CHAR(10)	Save file name
453	1C5	CHAR(10)	Save file library name
463	1CF	CHAR(17)	Save label
480	1E0	CHAR(8)	Save active date and time
488	1E8	CHAR(44)	Reserved

OBJL0600 List Data Section: The following information is returned in the list data section of the OBJL0600 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0500 format
532	214	CHAR(8)	Last-used date and time
540	21C	CHAR(8)	Reset date and time
548	224	BINARY(4)	Days-used count
552	228	CHAR(1)	Usage information updated
553	229	CHAR(23)	Reserved

OBJL0700 List Data Section: The following information is returned in the list data section of the OBJL0700 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0600 format
576	240	BINARY(4)	Object size
580	244	BINARY(4)	Object size multiplier
584	248	CHAR(1)	Object overflowed ASP indicator
585	249	CHAR(3)	Reserved

Field Descriptions

Allow change by program. A 1-character variable that is used to return the allow change by program flag. A 1 is returned if the object can be changed with the Change Object Description (QLICOBJD) API. A 0 is returned if the object cannot be changed with the API.

Authorized program analysis report (APAR). The identifier of the authorized program analysis report (APAR) that caused this object to be replaced. The field is blank if the object did not change because of an APAR.

Auxiliary storage pool. The auxiliary storage pool ID.

Changed by program. A 1-character variable that is used to return the changed by program flag. A 1 is returned if the object has been changed with the QLICOBJD API. A 0 is returned if the object has not been changed by the API.

Change date and time. The time at which the object was last changed, in system time-stamp format.

Compiler. The licensed program identifier, version number, release level, and modification level of the compiler. The field has a ppppppVvvRrrMmm format where:

ppppppp The licensed program identifier.
Vvv The character V is followed by a 2-character version number.
Rrr The character R is followed by a 2-character release level.
Mmm The character M is followed by a 2-character modification level.

The field is blank if you do not compile the program.

Creation date and time. The time at which the object was created, in system time-stamp format.

Refer to the information about the Materialize System Object (MATSOBJ) machine interface (MI) instruction in the *MI Functional Reference* for information about the system time-stamp format.

Creator's user profile. The name of the user that created the object.

Days-used count. The number of days the object was used. If the object does not have a last-used date, the count is 0.

Extended object attribute. The extended attribute of the object, such as a program or file type. Extended attributes further describe the object. For example, an object type of *PGM may have a value of RPG (RPG program) or CLP (CL program), and an object type of *FILE may have a value of PF (physical file), LF (logical file), DSPF (display file), SAVF (save file), and so on.

Format name. The format of the returned output.

Information status. Whether or not the QUSLOBJ API returns the requested information for this object. Possible values are:

blank The requested information is returned. No errors occurred.
A No information is returned because the job that called the QUSLOBJ API does not have adequate authority to the object.
D The requested information is returned. However, the object is damaged and should be re-created as soon as possible.
L No information is returned because the object is locked.

If the value of this field is A or L, your application should not use the other fields for the object. Only the object name, library, and type fields contain accurate data.

Last-used date and time. The date and time at which the object was last used, in system time-stamp format. If the object has no last-used date, the field contains hexadecimal zeros.

Licensed program. The name, release level, and modification level of the licensed program if the retrieved object is part of a licensed program. The 7-character name starts in character position 1, the version number starts in position 8, the release level starts in position 11, and the modification level starts in position 14. The field is blank if the retrieved object is not a part of a licensed program.

Object compression status. Whether the object is compressed or decompressed. The status is returned in a 1-character variable with one of these values:

Y Compressed.
N Permanently decompressed and compressible.
X Permanently decompressed and *not* compressible.
T Temporarily decompressed.
F Saved with storage freed; compression status cannot be determined.

Temporarily decompressed objects exist in both decompressed and compressed form. *Permanently* decompressed objects exist in decompressed form only. The system handles some decompression automatically, depending on the type of object, the operation performed on it, and its frequency of use. For an overview of object compression and decompression, see the *CL Programmer's Guide*. For details about how to explicitly compress and decompress objects, see the entries for these commands in the *CL Reference*: Compress Object (CPROBJ), Decompress Object (DCPOBJ), and Reclaim Temporary Storage (RCLTMPSTG).

Object domain. The domain that contains the object. The value is *U if the object is in the user domain, or *S if the object is in the system domain.

Object level. The object control level for the created object.

Object library name specified. The name of the object library as specified in the call to the API.

Object library name used. The name of the library containing the object.

Object name specified. The name of the object as specified in the call to the API.

Object name used. The name of the object.

Object overflowed ASP indicator. The 1-character variable that returns the object overflowed auxiliary storage pool (ASP) indicator. The value is 1 if the object overflowed the ASP in which it resides; the value is 0 if the object has not overflowed the ASP. For objects in the system ASP, a 0 is always returned because it is not possible for an object that resides in the system ASP to overflow its ASP.

Object owner. The name of the object owner's user profile.

Object restored date and time. The time at which the object was restored, in system time-stamp format. If the object has never been restored, the field contains hexadecimal zeros.

Object saved date and time. The time at which the object was saved, in system time-stamp format. If the object has never been saved, the field contains hexadecimal zeros.

Object size. The size of the object in units of the size multiplier. The object size is equal to or smaller than the object size multiplied by the object size multiplier.

Object size multiplier. The value to multiply the object size by to get the true size. The value is 1 if the object is smaller than 1 000 000 000 bytes, and 1024 if it is larger.

Object type specified. The type of the object as specified in the call to the API.

Object type used. The type of the object. For a list of all the available object types, see the *CL Reference*.

Program temporary fix (PTF). The number of the program temporary fix (PTF) number that caused this object to be replaced. This field is blank if the object was not changed because of a PTF.

Reserved. An ignored field.

Reset date and time. The date the days-used count was last reset to zero, in system time-stamp format. If the days-used count has never been reset, the field contains hexadecimal zeros.

Save active date and time. The date and time the object was last saved when the SAVACT(*LIB, *SYSDFN, or *YES) save operation was specified, in system time-stamp format. This parameter is found on the Save Library (SAVLIB), Save Object (SAVOBJ), Save Changed Object (SAVCHGOBJ), and Save Document Library Object (SAVDLO) CL commands. If the object has never been saved or if SAVACT(*NO) was specified on the last save operation for the object, the field contains hexadecimal zeros.

Save command. The command used to save the object. The field is blank if the object was not saved.

Save device. The type of device to which the object was last saved. The field is *SAVF if the last save operation was to a save file. The field is *DKT if the last save operation was to diskette. The field is *TAP if the last save operation was to tape. The field is blank if the object was not saved.

Save file library name. The name of the library that contains the save file if the object was saved to a save file. The field is blank if the object was not saved to a save file.

Save file name. The name of the save file if the object was saved to a save file. The field is blank if the object was not saved to a save file.

Save label. The file label used when the object was saved. The variable is blank if the object was not saved to tape or diskette. The value of the variable corresponds to the value specified for the LABEL parameter on the command used to save the object.

Save sequence number. The tape sequence number assigned when the object was saved on tape. The field contains zeros if the object was not saved.

Save size. The size of the object in bytes of storage at the time of the last save operation. The save size is equal to or smaller than the save size multiplied by the save size multiplier. The field contains zeros if the object was not saved.

Save size multiplier. The value to multiply the save size by to get the true size. The value is 1 if the save size is smaller than 1 000 000 000 bytes, and 1024 if it is larger.

Save volume ID. The tape or diskette volumes used for saving the object. The variable returns a maximum of ten 6-character volumes. The volume IDs begin in character positions 1, 8, 15, 22, 29, 36, 43, 50, 57, and 64. If more than 10 volumes are used, a 1 is returned in the 71st character of the variable; otherwise, the 71st character is blank. The field is blank if the object was last saved to a save file, or if it was never saved.

Source file library name. The name of the library that contains the source file used to create the object. The field is blank if no source file created the object.

Source file member name. The name of the member in the source file. The field is blank if no source file created the object.

Source file name. The name of the source file used to create the object. The field is blank if no source file created the object.

Source file updated date and time. The date and time the member in the source file was last updated. This field is in the CYYMMDDHHMMSS format where:

C Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
YY Year
MM Month
DD Day
HH Hour
MM Minute
SS Second

The field is blank if no source file created the object.

Storage. The storage status of the object data. *FREE indicates the object data is freed and the object is suspended. *KEEP indicates the object data is not freed and the object is not suspended.

System level. The level of the operating system when the object was created. The field has a VvvRrrMmm format where:

Vvv The character V is followed by a 2-character version number.
Rrr The character R is followed by a 2-character release level.
Mmm The character M is followed by a 2-character modification level.

System where object was created. The name of the system on which the object was created.

Text description. The text description of the object. The field is blank if no text description is specified.

Usage information updated. Whether the object usage information is updated for this object type. The indicator is returned as Y (Yes) or N (No).

User changed. Whether the user program was changed. A character 1 is returned if the user changed the object. If the object was not changed by the user, the field is character 0.

User-defined attribute. Further defines an object type. This field is set by the user while using the QLICOBJD API.

User space library name. The library containing the user space, as specified in the call to the API.

User space name. The name of the user space.

Error Messages

CPF3CAA E List is too large for user space &1.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C20 E Error found by program &1.
CPD3C21 D Format name &1 is not valid.
CPD3C31 D Object type &1 is not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C31 E Object type &1 is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9804 E Object &2 in library &3 damaged.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.

| CPF9812 E File &1 in library &2 not found.
| CPF9830 E Cannot assign library &1.
| CPF9838 E User profile storage limit exceeded.

Retrieve Object Description (QUSROBJD) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Object and library name	Input	Char(20)
5	Object type	Input	Char(10)

Optional Parameter:

6	Error code	I/O	Char(*)

The Retrieve Object Description (QUSROBJD) API lets you retrieve object information about a specific object. This information is similar to the information returned using the Display Object Description (DSPOBJD) command.

You can use the QUSROBJD API to:

- Determine who owns which objects in the specified libraries
- Provide disk management functions based on the object's size and use
- Provide backup analysis based on when the object was last saved or last updated
- Provide source member and object analysis from source member information to verify that the current source was used to create the specified object
- Work with a list of objects created by the QUSLOBJ API

Authorities and Locks

Library Authority *READ
Target Object Authority for Non-*FILE Objects
Any authority other than *EXCLUDE
Target Object Authority for *FILE Objects
*OBJOPR

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the requested information. It can be smaller than the format requested as long as the next parameter, length of receiver variable, specifies the length correctly. When this variable is smaller than the format, the API returns only the data that the variable can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. The minimum length is 8 bytes. Do not specify a length that is longer than the receiver variable; the results are unpredictable.

Format name
INPUT; CHAR(8)
The content and format of the information returned for each specified member. The possible format names are:

OBJD0100 Basic information (fastest)
OBJD0200 Information similar to that displayed by the programming development manager (PDM)
OBJD0300 Service information
OBJD0400 Full information (slowest)

These are described in the following sections.

Object and library name
INPUT; CHAR(20)
The object for which you want to retrieve information, and the library in which it is located. The first 10 characters contain the object name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Object type
INPUT; CHAR(10)
The type of object for which you want to retrieve the information. You can only specify external object types. Refer to the *CL Reference* for a complete list of available object types.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

OBJD0100 Format

The following information is returned for the OBJD0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 33-10.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Object name
18	12	CHAR(10)	Object library name
28	1C	CHAR(10)	Object type
38	26	CHAR(10)	Return library
48	30	BINARY(4)	Auxiliary storage pool
52	34	CHAR(10)	Object owner

Offset		Type	Field
Dec	Hex		
62	3E	CHAR(2)	Object domain
64	40	CHAR(13)	Creation date and time
77	4D	CHAR(13)	Object change date and time

OBJD0200 Format

The following information is returned for the OBJD0200 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJD0100 format
90	5A	CHAR(10)	Extended object attribute
100	64	CHAR(50)	Text description
150	96	CHAR(10)	Source file name
160	A0	CHAR(10)	Source file library name
170	AA	CHAR(10)	Source file member name

OBJD0300 Format

The following information is returned for the OBJD0300 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJD0200 format
180	B4	CHAR(13)	Source file updated date and time
193	C1	CHAR(13)	Object saved date and time
206	CE	CHAR(13)	Object restored date and time
219	DB	CHAR(10)	Creator's user profile
229	E5	CHAR(8)	System where object was created
237	ED	CHAR(7)	Reset date
244	F4	BINARY(4)	Save size
248	F8	BINARY(4)	Save sequence number
252	FC	CHAR(10)	Storage
262	106	CHAR(10)	Save command
272	110	CHAR(71)	Save volume ID
343	157	CHAR(10)	Save device
353	161	CHAR(10)	Save file name
363	16B	CHAR(10)	Save file library name
373	175	CHAR(17)	Save label
390	186	CHAR(9)	System level
399	18F	CHAR(16)	Compiler
415	19F	CHAR(8)	Object level
423	1A7	CHAR(1)	User changed
424	1A8	CHAR(16)	Licensed program
440	1B8	CHAR(10)	Program temporary fix (PTF)
450	1C2	CHAR(10)	Authorized program analysis report (APAR)

OBJD0400 Format

The following information is returned for the OBJD0400 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJD0300 format
460	1CC	CHAR(7)	Last-used date
467	1D3	CHAR(1)	Usage information updated
468	1D4	BINARY(4)	Days-used count
472	1D8	BINARY(4)	Object size
476	1DC	BINARY(4)	Object size multiplier
480	1E0	CHAR(1)	Object compression status
481	1E1	CHAR(1)	Allow change by program
482	1E2	CHAR(1)	Changed by program
483	1E3	CHAR(10)	User-defined attribute
484	1E4	CHAR(1)	Object overflowed ASP indicator
494	1EE	CHAR(13)	Save active date and time

Field Descriptions

Allow change by program. A 1-character variable that is used to return the allow change by program flag. A 1 is returned if the object can be changed with the Change Object Description (QLICOBJD) API. A 0 is returned if the object cannot be changed with the API.

Authorized program analysis report (APAR). The identifier of the authorized program analysis report (APAR) that caused this object to be replaced. The field is blank if the object did not change because of an APAR.

Auxiliary storage pool. The auxiliary storage pool ID.

Bytes available. The length of all data available to return. All available data is returned if enough space is provided.

Bytes returned. The length of the data actually returned.

Changed by program. A 1-character variable that is used to return the changed by program flag. A 1 is returned if the object has been changed with the QLICOBJD API. A 0 is returned if the object has not been changed by the API.

Compiler. The licensed program identifier, version number, release level, and modification level of the compiler. The field has a ppppppVvvRrrMmm format where:

ppppppp	The licensed program identifier.
Vvv	The character V is followed by a 2-character version number.
Rrr	The character R is followed by a 2-character release level.
Mmm	The character M is followed by a 2-character modification level.

The field is blank if you do not compile the program.

Creation date and time. The date and time the object was created. The creation date and time field is in the CYYMMDDHHMMSS format where:

C	Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
YY	Year
MM	Month
DD	Day
HH	Hour
MM	Minute
SS	Second

Creator's user profile. The name of the user that created the object.

Days-used count. The number of days the object was used. If the object does not have a last used date, the count is 0.

Extended object attribute. The extended attribute of the object, such as a program or file type. Extended attributes further describe the object. For example, an object type of *PGM may have a value of RPG (RPG program) or CLP (CL program), and an object type of *FILE may have a value of PF (physical file), LF (logical file), DSPF (display file), SAVF (save file), and so on.

Last-used date. The date the object was last used. This field is in the CYYMMDD format, which is the same format used for the reset date. If the object has no last-used date, the field is blank.

Licensed program. The name, release level, and modification level of the licensed program if the retrieved object is part of a licensed program. The 7-character name starts in character position 1, the version number starts in position 8, the release level starts in position 11, and the modification level starts in position 14. The field is blank if the retrieved object is not a part of a licensed program.

Object change date and time. The date and time the object was last changed. The format is the same as the creation date description, or it is blank if the object was not changed.

Object compression status. Whether the object is compressed or decompressed. The status is returned in a 1-character variable with one of these values:

Y	Compressed.
N	Permanently decompressed and compressible.
X	Permanently decompressed and *not* compressible.
T	Temporarily decompressed.
F	Saved with storage freed; compression status cannot be determined.

Temporarily decompressed objects exist in both decompressed and compressed form. *Permanently* decompressed objects exist in decompressed form only. The system handles some decompression automatically, depending on the type of object, the operation performed on it, and its frequency of use. For an overview of object compression and decompression, see the *CL Programmer's Guide*. For details about how to explicitly compress and decompress objects, see the entries for these commands in the *CL Reference*: Compress Object (CPROBJ), Decompress Object (DCPOBJ), and Reclaim Temporary Storage (RCLTMPSTG).

Object domain. The domain that contains the object. The value is *U if the object is in the user domain, or *S if the object is in the system domain.

Object level. The object control level for the created object.

Object library name. The name of the library containing the object.

Object name. The name of the object.

Object overflowed ASP indicator. The 1-character variable that returns the object overflowed auxiliary storage pool (ASP) indicator. The value is 1 if the object overflowed the ASP in which it resides; the value is 0 if the object has not overflowed the ASP. For objects in the system ASP, a 0 is always returned because it is not possible for an object that resides in the system ASP to overflow its ASP.

Object owner. The name of the object owner's user profile.

Object restored date and time. The date and time the object was last restored. The format is the same as for the creation date, or it is blank if the object was never restored.

Object saved date and time. The date and time the object was last saved. The format is the same as for the creation date description, or it is blank if the object was never saved.

Object size. The size of the object in units of the size multiplier. The object size is equal to or smaller than the object size multiplied by the object size multiplier.

Object size multiplier. The value to multiply the object size by to get the true size. The value is 1 if the object is smaller than 1 000 000 000 bytes, and 1024 if it is larger.

Object type. The object type. For a list of all the available object types, see the *CL Reference*.

Program temporary fix (PTF). The number of the program temporary fix (PTF) number that caused this object to be replaced. This field is blank if the object was not changed because of a PTF.

Reset date. The date the days-used count was last reset to 0. The reset date field is in the CYYMMDD format, where:

C	Century
YY	Year
MM	Month
DD	Day

If the days-used count was not reset, the date is blank.

Return library. The name of the library containing the object if *LIBL or *CURLIB is specified for the library name on the object parameter.

Save active date and time. The date and time the object was last saved when the SAVACT(*LIB, *SYSDFN, or *YES) save operation was specified, in system time-stamp format. This parameter is found on the Save Library (SAVLIB), Save Object (SAVOBJ), Save Changed Object (SAVCHGOBJ), and Save Document Library Object (SAVDLO) CL commands. The format is the same as for the creation date description, or it is blank if the object was never saved or if SAVACT(*NO) was specified on the last save operation for the object.

Save command. The command used to save the object. The field is blank if the object was not saved.

Save device. The type of device to which the object was last saved. The field is *SAVF if the last save operation was to a save file. The field is *DKT if the last save operation was to diskette. The field is *TAP if the last save operation was to tape. The field is blank if the object was not saved.

Save file library name. The name of the library that contains the save file if the object was saved to a save file. The field is blank if the object was not saved to a save file.

Save file name. The name of the save file if the object was saved to a save file. The field is blank if the object was not saved to a save file.

Save label. The file label used when the object was saved. The variable is blank if the object was not saved to tape or diskette. The value of the variable corresponds to the value specified for the LABEL parameter on the command used to save the object.

Save sequence number. The tape sequence number assigned when the object was saved on tape. The field contains zeros if the object was not saved.

Save size. The size of the object in bytes of storage at the time of the last save operation. The field contains zeros if the object was not saved.

Save volume ID. The tape or diskette volumes used for saving the object. The variable returns a maximum of 10 six-character volumes. The volume IDs begin in character positions 1, 8, 15, 22, 29, 36, 43, 50, 57, and 64. If more than 10 volumes are used, a '1' is returned in the 71st character of the variable; otherwise, the 71st character is blank. The field is blank if the object was last saved to a save file, or if it was never saved.

Source file library name. The name of the library that contains the source file used to create the object. The field is blank if no source file created the object.

Source file member name. The name of the member in the source file. The field is blank if no source file created the object.

Source file name. The name of the source file used to create the object. The field is blank if no source file created the object.

Source file updated date and time. The date and time the member in the source file was last updated. The field is in the same format as the creation time and date. The field is blank if no source file created the object.

Storage. The storage status of the object data. *FREE indicates the object data is freed and the object is suspended. *KEEP indicates the object data is not freed and the object is not suspended.

System level. The level of the operating system when the object was created. The field has a VvvRrrMmm format where:

Vvv	The character V is followed by a 2-character version number.
Rrr	The character R is followed by a 2-character release level.
Mmm	The character M is followed by a 2-character modification level.

System where object was created. The name of the system on which the object was created.

Text description. The text description of the object. The field is blank if no text description is specified.

Usage information updated. Whether the object usage information is updated for this object type. The indicator is returned as Y (Yes) or N (No).

User changed. Whether the user program was changed. A character 1 is returned if the user changed the object. If the object was not changed by the user, the field is character 0.

User-defined attribute. Further defines an object type. This field is set by the user while using the QLICOBJD API.

Error Messages

CPF2101 E Object type *&1 not valid.
CPF2150 E Object information function failed.
CPF2151 E Operation failed for &2 in &1 type *&3.

| CPF2451 E Message queue &1 is allocated to another job.
| CPF3CF1 E Error code parameter not valid.
| CPF3CF2 E Error(s) occurred during running of &1 API.
| CPF3C07 E Error occurred while retrieving information from object &1.
| CPD3C20 D Error occurred with receiver variable specified.
| CPD3C21 D Format name &1 is not valid.
| CPD3C24 D Length of the receiver variable is not valid.
| CPD3C31 D Object type &1 is not valid.
| CPF3C19 E Error occurred with receiver variable specified.
| CPF3C21 E Format name &1 is not valid.
| CPF3C24 E Length of the receiver variable is not valid.
| CPF3C31 E Object type &1 is not valid.
| CPF3C36 E Number of parameters, &1, entered for this API was not valid.
| CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
| CPF87A9 E Device with RMTLOCNAME &1 not found.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9811 E Program &1 in library &2 not found.
| CPF9812 E File &1 in library &2 not found.
| CPF9814 E Device &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9821 E Not authorized to program &1 in library &2.
| CPF9822 E Not authorized to file &1 in library &2.
| CPF9825 E Not authorized to device &1.
| CPF9830 E Cannot assign library &1.
| CPF9831 E Cannot assign device &1.

Part 16. Virtual Terminal APIs

Chapter 34. Introduction to Virtual Terminal APIs

The virtual terminal APIs allow your AS/400 application programs to interact with AS/400 application programs that are performing work station input and output (I/O).

A **virtual terminal** is a device that does not have hardware associated with it. It forms a connection between your application and AS/400 applications, representing a physical work station (possibly on a remote system). The OS/400 licensed program manages the virtual terminal, which directs work station I/O performed by an AS/400 application to the virtual terminal. The virtual terminal APIs allow another AS/400 application, called a **server program**, to work with the data associated with the virtual terminal.

In a distributed systems environment, the requesting program is called a **client**; the answering program is called a **server**. The client and server programs may reside on the same AS/400 system or may be distributed between two different systems. The server program generally runs on behalf of (or in conjunction with) the client program. Together, the server program and the client program allow a work station to be supported as if the work station were connected locally.

Chapter 35, "Virtual Terminal APIs" provides the virtual terminal APIs, including syntax, parameters, notes about usage, and associated AS/400 system messages. Chapter 36, "Virtual Terminal Run-Time Example" contains an example of how the OS/400 operating system, the server program, the client program, and the work station device interact when processing a system request.

Distributed 5250 Emulation Model

Figure 34-1 shows a model for a distributed systems environment between an IBM PS/2* work station and an AS/400 system. The client program resides on the PS/2 work station, while the server program resides on the AS/400 system. This model is similar to the way the AS/400 supports a PC Support/400 work station function.

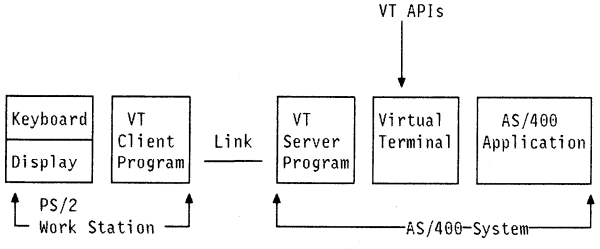


Figure 34-1. Example Virtual Terminal Client/Server Model

The client program running on the PS/2 work station shown in Figure 34-1 does the following:

- Accepts data from the server program and displays the data on the PS/2 display
- Accepts data from the PS/2 keyboard
- Converts the data from the format required by the PS/2 display and keyboard to the format required by the server program (5250 data stream)
- Sends the data to the server program on the AS/400 system

The link between the client program and server program uses DOS and OS/400 communications support. This may be LU 6.2, TCP/IP, or some other communications protocol.

You can write a server application program in any AS/400-supported high-level language (HLL), such as the C/400 programming language. The server program provides work station support for the AS/400 system acting as the server. The virtual terminal APIs allow a server program to read and write virtual terminal data. Virtual terminal data is always in a 5250 data stream format. See the *5250 Functions Reference Manual*, SA21-9247, for more information on 5250 data streams.

The virtual terminal APIs provide an interface between the server program and the virtual terminal supported by the OS/400 licensed program. The virtual terminal represents the OS/400 link between the server program and the AS/400 application and is managed entirely by the OS/400 licensed program.

The AS/400 application is a licensed program or a user-written application that performs a standard data transfer to a work station. This application can be written in any AS/400-supported HLL.

AS/400 Job Information

Several AS/400 jobs are involved when you use the virtual terminal APIs. The jobs can be classified into two groups:

- Server program jobs
- Application jobs

Each group may consist of one or more jobs, depending on the way the server program is written and the way the AS/400 applications are being run.

The server program can be written to run in a single job or in more than one job, depending on the number of work stations to be supported per job. For example, you can support each work station using a single job by routing all requests from the work station client program to a particular server program job.

AS/400 Subsystem Information

A server program should run in the same subsystem that other server programs are running. For controlling resource use, such as main storage, a separate subsystem for running all server programs is usually best. This is also advantageous for allowing performance tuning, such as the number of page faults. Generally, AS/400 applications run in subsystem QBASE.

If the subsystem under which you want the server program to run was created with the system defaults, you will not have to add a work station entry to the subsystem description. However, if you do need to add a work station entry to the subsystem description, you can use the Add Work Station Entry (ADDWSE) command.

Before a work station is allowed to sign-on, it must be defined to a subsystem. In this case, the work station is the virtual terminal device (QPADEVnnnn) automatically created by the OS/400 licensed program. The work station name, work station type, or *ALL must be specified in the subsystem description. Use the Display Subsystem Description (DSPSBSD) command to see the list of work station entries defined for a subsystem. The following command can be used to add all work station types to a subsystem named QBASE:

```
ADDWSE SBSD(QBASE) WRKSTNTYPE(*ALL)
```

For more information on automatically creating virtual terminals, see "Step 1: Setting the Number of Automatically Created Virtual Terminals" on page 34-3.

Note: The ADDWSE command is only valid when the subsystem description is not active.

Work Station Types

You must specify a work station type when a virtual terminal device is opened. A server program can select several different types of work stations. See "Supported Work Station Types and Models" on page 35-2 for a list of the types of work stations that are supported.

Data Queues

The OS/400 licensed program uses data queues to send data to the server program. The server program also uses data queues for interprocess communications with other AS/400 applications and API calls.

The data queue must exist before your application uses the virtual terminal APIs to open a path to a virtual terminal. See the *CL Reference* for details about creating and deleting data queues. See "Open Virtual Terminal Path (QTVOPNVT) API" on page 35-1 for information on how a server program specifies the name of the data queue to be used.

The OS/400 licensed program communicates special events to the server program using the data queue. The AS/400 system provides the information about the special events using a data queue entry.

The following events result in data queue entries being sent:

- The virtual terminal receives data from an AS/400 application
- OS/400 closes the virtual terminal (This occurs when an AS/400 application's job is canceled.)

Figure 34-2 shows the structure of OS/400 data queue entries that are sent to the data queue when an application uses the virtual terminal APIs. All data queue entries have the same format.

Figure 34-2. Format for OS/400 Data Queue Entries

Name	Type	Description
Entry Type	CHAR(10)	Always set by OS/400 to *VRTTRM.
Entry ID	CHAR(2)	Entry ID associated with entry. Valid values for the 2 characters in this parameter are: 1. OS/400 is closing (terminating) the session with the virtual terminal. The server program should perform a close to indicate that the server program is done using the virtual terminal. 2. An AS/400 application has sent data to the virtual terminal. see "Read from Virtual Terminal (QTVRDVT) API" on page 35-3 for information on how to read the data. **Note**: All other values are reserved by AS/400.
VTHandle	CHAR(16)	The virtual terminal handle associated with the virtual terminal communications path previously opened by calling the Open Virtual Terminal Path (QTVOPNVT) API. See "Open Virtual Terminal Path (QTVOPNVT) API" on page 35-1 for additional details.
	CHAR(52)	Reserved
Key	CHAR(*)	Key value specified when the virtual terminal communications path was opened. See "Open Virtual Terminal Path (QTVOPNVT) API" on page 35-1 for additional details on specifying the key value. The key value can be used for retrieving data queues by key.

Preparing to Use the Virtual Terminal APIs

The following steps are required to prepare your AS/400 system to run an application using the virtual terminal APIs:

1. Set the number of automatically created virtual terminals using the Automatic virtual device configuration indicator (QAUTOVRT) system value
2. Set the Limit security officer device access (QLMTSECOFR) system value
3. Create user profiles using the Create User Profile (CRTUSRPRF) command

Step 1: Setting the Number of Automatically Created Virtual Terminals

The OS/400 licensed program uses virtual terminals to allow a server program to interact with its client by sending and receiving data with AS/400 applications. The OS/400 operating system will automatically select (and create if necessary) these virtual terminals for you.

The QAUTOVRT system value specifies the maximum number of terminals that will be automatically configured by the system. When you set the QAUTOVRT system value, the OS/400 licensed program automatically configures the required virtual controllers and terminals. Controllers coordinate and control the operation of one or more input/output terminals (such as work stations) and synchronize the operation of such terminals with the operation of the entire system. Use the Change System Value (CHGSYSVAL) command to change the value of the QAUTOVRT system value. For example, entering the following command string changes the number of virtual terminals that can be allocated on a system to 50:

```
CHGSYSVAL SYSVAL(QAUTOVRT) VALUE(50)
```

To determine and set the maximum number of users you want signed on to the AS/400 system at any time, do the following:

- Set the QAUTOVRT system value to 9999, the maximum value allowed.
- Have your users use the AS/400 system until you decide that the number of virtual terminals created is sufficient for normal system operation.
- Use the Work with Configuration Status (WRKCFGSTS) command to determine the number of work stations configured.
- Change the QAUTOVRT system value from 9999 to the number of virtual terminals you require for normal operation.

If you have never allowed virtual terminals to be configured automatically on your system, the QAUTOVRT system value is 0. As a result, you cannot use the virtual terminal APIs because the OS/400 licensed program is not able to create more work stations than the number specified. If you change the QAUTOVRT system value to 10, the next virtual terminal path opened causes the OS/400 licensed program to create a virtual terminal. This virtual terminal is created because the number of virtual terminals on the controller (0) is less than the number specified in the QAUTOVRT system value (10). Even if you change the specified number to 0 again, the next virtual terminal opened may succeed if a virtual terminal exists that is not being used.

If a virtual terminal does not exist or is in use, the OS/400 licensed program does not create a new virtual terminal because the number of virtual terminals currently existing is greater than or equal to the specified QAUTOVRT system value. When the number of virtual terminals that currently exist is greater than or equal to the QAUTOVRT system value, the message CPF8940, "Cannot automatically select virtual device", is sent to the system operator message queue (QSYSOPR). You must either try again when a virtual terminal description becomes available or increase the QAUTOVRT system value.

The OS/400 operating system uses the following conventions for naming virtual controllers and work stations:

- Virtual controllers are named QPACTL*nn*
- Virtual terminal descriptions are named QPADEV*xxxx*

Consider the following when you allow the OS/400 licensed program to automatically configure work stations:

- The OS/400 licensed program does not delete virtual terminals, even when the number of work stations attached to virtual controllers exceeds the limit set by QAUTOVRT.

  If you want the extra work stations deleted, you must manually delete them.
- The OS/400 licensed program allows a maximum of 250 virtual terminals on the QPACTL01 controller before it creates QPACTL02. This value is usually adequate. If you delete work stations to enforce a smaller value for the QAUTOVRT limit, begin by deleting the work stations from the controller with the highest numeric value in its name (where *nn* in the QPACTL*nn* name is largest).

  Note: Changing this system value affects other AS/400 products and programs requiring automatic configuration. This includes TCP/IP TELNET, 5250 display station pass-through, and any other programs using the virtual terminal APIs.

Step 2: Setting the Limit Security Officer (QLMTSECOFR) Value

The Limit security officer device access (QLMTSECOFR) system value, limits the devices the security officer can sign on to. The security officer controls all of the security authorizations provided by the AS/400 system. If the QLMTSECOFR value is greater than zero, the security officer must be authorized to use the virtual device descriptions. However, when this value equals 0, the system does not limit the devices the security officer can use to sign on the system.

When the system security level (QSECURITY) system value is set to 30, a security officer with all object authority (*ALLOBJ) must be authorized to use the work stations. For example, for each display station that a security officer wants to sign on to (local, remote, or virtual), the user must authorize the security officer using the following Grant Object Authority (GRTOBJAUT) command:

```
GRTOBJAUT OBJ(display-name) OBJTYPE(*DEVD) AUT(*CHANGE) USER(QSECOFR)
```

This procedure is very important because using the virtual terminal APIs automatically configures virtual terminals (devices). Automatic configuration is a function that names and creates the descriptions of network devices and controllers attached to a line. If the QLMTSECOFR value is set to 0, all virtual terminals automatically configured when you use the virtual terminal APIs can be used by the security officer. If you set the QLMTSECOFR value to 1, your security officer is not able to use the virtual terminals unless you specifically grant object authority to the

security officer for that virtual terminal. The automatic configuration support can delete and re-create the virtual terminal. If this occurs, authority must be granted to the security officer each time the virtual terminal is created.

Security Considerations: The number of sign-on attempts allowed increases if virtual terminals are automatically configured. The number of sign-on attempts is equal to the number of system sign-on attempts allowed multiplied by the number of virtual terminals that can be created. The number of system sign-on attempts allowed is defined by the QMAXSIGN system value. The number of virtual terminals that can be created is defined by the QAUTOVRT system value.

Step 3: Creating User Profiles

You should create one or more user profiles on the AS/400 system for users of the virtual terminal supported by the client and server programs. The default user profile is *SYS. The following example shows a sample user profile:

```
CRTUSRPRF  USRPRF(CLERK1)  PASSWORD(unique-password)
           JOBD(CLERKLIB/CLERKL1)
           TEXT('User profile for one group of clerks')
```

Creating Your Own Virtual Controllers and Devices

You can create your own virtual controllers and devices (terminals); however, you must use the same naming conventions as the automatic controller and device creation support. You may want to create the virtual terminal descriptions to control the number of sign-on attempts possible by not allowing automatic configuration of virtual terminals (which allows additional sign-on attempts to occur). See "Security Considerations" for additional information.

If you do not want to use automatically created descriptions, do the following:

- Use the Create Controller Description (Virtual Work Station) (CRTCTLVWS) command to create a controller description for a virtual terminal.

  ```
  CRTCTLVWS CTLD(QPACTL01)
            TEXT('Virtual Controller for virtual terminals')
  ```

  Note: You must use the OS/400 naming convention, QPACTL*nn*, for naming virtual controllers, where *nn* is a decimal number starting at 01.

- Use the Create Device Description (Display) (CRTDEVDSP) command to create a virtual terminal as follows:

  ```
  CRTDEVDSP DEVD(QPADEV0001) DEVCLS(*VRT)
            TYPE(5251) MODEL(11) CTL(QPACTL01)
            TEXT('24 X 80 Monochrome
               Display for Server Program')
  ```

- The OS/400 licensed program automatically varies on the controller and terminal that you have created. You must use the OS/400 naming convention, QPADEV*nnnn*, for naming virtual device descriptions, where *nnnn* is a decimal number starting at 0001.

After creating the descriptions, you must authorize the server program to use them. Use the Grant Object Authority (GRTOBJAUT) command to authorize the user profile used by the server program to the descriptions. This can be done using the following commands:

```
GRTOBJAUT OBJ(QPACTL01) OBJTYPE(*CTLD)
          AUT(*CHANGE) USER(user-ID)

GRTOBJAUT OBJ(QPADEV0001) OBJTYPE(*DEVD)
          AUT(*CHANGE) USER(user-ID)
```

You may want to prevent virtual terminals from being created automatically. To do this, set the QAUTOVRT system value to 0 as follows:

```
CHGSYSVAL SYSVAL(QAUTOVRT) VALUE(0)
```

See "Step 1: Setting the Number of Automatically Created Virtual Terminals" on page 34-3 for additional information.

Note: Changing this system value affects other AS/400 products and programs requiring automatic configuration. This includes TELNET, 5250 display station pass-through, and any other programs using the virtual terminal APIs.

Developing Client and Server Programs

When developing client and server programs using the virtual terminal APIs, you need to take the following items into consideration:

- The client program should be able to:
  - Interrupt the server program
  - Check the server program's status
  - Discard data from the AS/400 application
- The user should be able to configure a time-out to be used by the client program while waiting for screens from the server program.
- Pressing the Print key on the work station should create a file to be printed at either the work station or the AS/400 printer.

Chapter 35. Virtual Terminal APIs

This chapter describes the syntax and parameters for the virtual terminal APIs. The virtual terminal path APIs consist of the program calls shown in the following list:

Close Virtual Terminal Path (QTVCLOVT) closes one or all open virtual terminal paths.
Open Virtual Terminal Path (QTVOPNVT) opens a path to a virtual terminal.
Read from Virtual Terminal (QTVRDVT) reads data from the virtual terminal into a data buffer.
Send Request for OS/400 Function (QTVSNDRQ) sends a request to perform a particular function.
Write to Virtual Terminal (QTVWRTVT) writes data to a virtual terminal from a data buffer.

The APIs are presented in alphabetical order.

Close Virtual Terminal Path (QTVCLOVT) API

Parameters

Required Parameter Group:

1	Virtual terminal handle	Input	Char(16)
2	Error code	I/O	Char(*)

The Close Virtual Terminal Path (QTVCLOVT) API closes one or all open virtual terminal paths. To close all open virtual terminal paths, the handle must be set to zero.

Required Parameter Group

Virtual terminal handle
INPUT; CHAR(16)
The reference code for the open virtual terminal path, created with the Open Virtual Terminal Path (QTVOPNVT) API. If this parameter is set to zero, all open virtual terminal paths are closed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.

Open Virtual Terminal Path (QTVOPNVT) API

Parameters

Required Parameter Group:

1	Virtual terminal handle	Output	Char(16)
2	Keyboard language type	Input	Char(3)
3	Character set	Input	Binary(4)
4	Code page	Input	Binary(4)
5	Work station type	Input	Binary(4)
6	Data queue name and library	Input	Char(20)
7	Key value	Input	Char(*)
8	Key value length	Input	Binary(4)
9	Error code	I/O	Char(*)

The Open Virtual Terminal Path (QTVOPNVT) API opens a path to a virtual terminal, allowing a server program to interact with an AS/400 application. The virtual terminal path remains open until it is explicitly closed or the job is ended.

When you call the QTVOPNVT API, the operating system selects or automatically configures a virtual terminal for you and indicates that the device is logically turned on. The operating system then creates a sign-on display at the virtual terminal and sends a message to the specified data queue to signal the server program that data is available.

Authorities and Locks

Library Authority *USE
User Queue Authority *CHANGE
User Queue Lock *EXCLRD

Required Parameter Group

Virtual terminal handle
OUTPUT; CHAR(16)
A reference code created by the OS/400 operating system to identify this open virtual terminal path in later calls to other virtual terminal APIs.

Keyboard language type
INPUT; CHAR(3)
The keyboard language type for the virtual terminal. To use the system value, specify blanks for this parameter. For a list of other valid values, see the Create Device Description (CRTDEVDSP) command in the *CL Reference*. For details about supported languages, see the *National Language Support Planning Guide*.

Character set
INPUT; BINARY(4)
The graphic character set for the virtual terminal. Valid values are a specific graphic character set number and these special values:

0 The graphic character set system value is used.

-1 The keyboard language type is used to select the appropriate character set.

For details about the graphic character sets you can specify, see the *National Language Support Planning Guide*.

Note: The graphic character set system value is obtained from the default graphic character set and code page (QCHRID) system value.

Code page
INPUT; BINARY(4)
The code page for the virtual terminal. For details about the code pages you can specify, see the *National Language Support Planning Guide*. If you specified 0 or -1 for the character set parameter, you do not have to specify this parameter. When you use 0 for the character set parameter, the system value is used for this parameter. When you use -1 for the character set parameter, the code page value is derived from the keyboard language type parameter.

Note: The code page system value is obtained from the default graphic character set and code page (QCHRID) system value.

Work station type
INPUT; BINARY(4)
The type of work station to use. Valid values are 1 through 15. See "Supported Work Station Types and Models" for an explanation of the values.

Other work station types and models are supported. You can specify these by determining their equivalents in Figure 35-1. For a more detailed list of work station types and equivalents, see "Supported Work Station Types and Models."

If a virtual terminal description does not yet exist for the work station type specified, the operating system attempts to configure the work station automatically. Automatic configuration is controlled by the Automatic virtual device configuration (QAUTOVRT) system value, which specifies the number of virtual terminals that the operating system can configure automatically. See "Step 1: Setting the Number of Automatically Created Virtual Terminals" on page 34-3 for more information on the QAUTOVRT value.

Data queue name and library
INPUT; CHAR(20)
The name and library of the data queue used by your application program to receive data from the operating system asynchronously. The first 10 bytes are for the data queue name, and the second 10 bytes are for the library name. Allowable special values are:

***CURLIB** The job's current library
***LIBL** The library list

Key value
INPUT; CHAR(*)
The key value to use for the OS/400 data queue.

Key value length
INPUT; BINARY(4)
The length of the key value. Valid values are 0 through 256. If you specify 0, no key is used for data queue entries.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Supported Work Station Types and Models

Figure 35-1 details the values you can specify for the QTVOPNVT API's work station type parameter:

Figure 35-1 (Page 1 of 2). Work Station Types and Models

Value	Work Station Type and Model	Equivalent Type and Model	Description
1	5251 11		24 x 80 monochrome display.
2	5291 1	5291 2	24 x 80 monochrome display.
3	5292 2		24 x 80 color graphics display. This type is also emulated by a graphics work station feature.
4	5555 B01	5555 E01	24 x 80 monochrome double-byte character set (DBCS) display. This type is emulated by a monochrome work station feature that supports a DBCS display.
5	3196 A1	3196 A2 3196 B1 3196 B2 3476 EA	24 x 80 monochrome display. This type is emulated by a monochrome work station feature. This is what the ASCII devices emulate.

Figure 35-1 (Page 2 of 2). Work Station Types and Models

Value	Work Station Type and Model	Equivalent Type and Model	Description
6	3179 2	3197 C1 3197 C2 3476 EC 5292 1	24 x 80 color display. This type is emulated by a color work station feature.
7	3180 2	3197 D1 3197 D2 3197 W1 3197 W2	27 x 132 monochrome display.
8	3477 FC		27 x 132 wide-screen color display.
9	3477 FG	3477 FA 3477 FD 3477 FE 3477 FW	27 x 132 wide-screen monochrome display.
10	5555 C01	5555 F01	24 x 80 color double-byte character set (DBCS) display. This type is emulated by a color work station feature that supports a DBCS display.
11	5555 G01		24 x 80 double-byte character set (DBCS) monochrome graphics display. This type is emulated by a monochrome work station feature that supports a DBCS display.
12	5555 G02		24 x 80 color double-byte character set (DBCS) graphics display. This type is emulated by a color graphics work station feature that supports a DBCS display.
13	3486 BA		24 x 80 monochrome display.
14	3487 HA	3487 HG 3487 HW	24 x 80 or 27 x 132 monochrome display.
15	3487 HC		24 x 80 or 27 x 132 color display.

Notes:

1. All 5250 work stations, except 5555 Model B01, can operate as 5251 Model 11 work stations.
2. Selecting double-byte character set (DBCS) work stations requires the AS/400 primary language to be one of the DBCS national language versions (NLVs).

Error Messages

CPF1842 E Cannot access system value &1.
CPF3CF1 E Error code parameter not valid.
CPF87FA E Character identifier not valid.
CPF87F0 E Virtual terminal type value &1 not valid.
CPF87F1 E Queue key length &1 not valid.
CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.
CPF87F9 E Keyboard language type &1 not valid.

Read from Virtual Terminal (QTVRDVT) API

Parameters

Required Parameter Group:

1	Virtual terminal handle	Input	Char(16)
2	Read information	Output	Char(10)
3	Data buffer	Output	Char(*)
4	Number of bytes to read	Input	Binary(4)
5	Data received	Output	Binary(4)
6	Error code	I/O	Char(*)

The Read from Virtual Terminal (QTVRDVT) API reads data from the virtual terminal into the server program's data buffer. Your application should read data only if it has received an asynchronous notification message on the data queue, or if the more data flag was set on a previous read operation. The data received is in 5250 data stream format.

Only one full-screen display of data can be received at a time. If the data buffer is too small, partial displays are received and the more data flag for the QTVRDVT API's read information parameter is set to 1.

Before working with 5250 data streams, be sure to see the *IBM 5250 Information Display System Functions Reference Manual*.

Required Parameter Group

Virtual terminal handle
INPUT; CHAR(16)
The reference code for the open virtual terminal path, created by the operating system with the Open Virtual Terminal Path (QTVOPNVT) API.

Read information
OUTPUT; CHAR(10)
Information about the read operation. The characters and their meanings are:

1 The operation code, which gives the server program additional information about OS/400 status and what is expected of the server program. Valid values for this parameter are:

- **1** Invite
- **2** Output only
- **3** Put/get
- **4** Save display
- **5** Restore display
- **6** Read immediate
- **8** Read display
- **A** Cancel invite
- **B** Turn on message light
- **C** Turn off message light

For detailed descriptions of these codes, see "Read Operation Codes."

2 More data flag. Valid values for this parameter are:

- **0** There is no more data.
- **1** More data is available. Issue the read operation again to receive the additional data. This flag is set if the buffer specified on input is not large enough to hold all of the data received from the virtual terminal.

3 Key flag. Valid character values for this parameter are:

- **0** The Enter key was pressed.
- **1** The System Request key was pressed.

4-10 Reserved. These characters must be blank.

Data buffer
OUTPUT; CHAR(*)
The server program's buffer for receiving data from the virtual terminal. The data is a 5250 data stream.

The QTVRDVT API does not lock the data buffer. Thus, other applications should not use the buffer while the API is using it.

The data buffer should be large enough to hold the largest display of data expected. If it is not large enough for all the data to fit, the more data flag of the read information parameter is set to 1. Additional read requests must be performed, until all the remaining data is received and the more data flag is set back to 0.

Number of bytes to read
INPUT; BINARY(4)
The number of bytes to read from the data buffer. This number must be smaller than or equal to the size of the data buffer.

Data received
OUTPUT; BINARY(4)
The amount of data received from the virtual terminal in bytes. If no data is received from the virtual terminal, 0 is returned. Some read operations do not return any data.

For graphic work stations, a maximum of 24 576 (24KB) bytes of data can be returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Read Operation Codes

Figure 35-2 describes the operation codes that can be returned on a read request.

Figure 35-2 (Page 1 of 2). Read Operation Codes

Value	Response Name	Description
1	Invite	No data is returned from this read request. The operating system and the application are ready to receive data. The server program is expected to follow this with a write operation when data becomes available from the client program.
2	Output only	This read request returned some data. The server program should send the data to the client program. However, the operating system is not ready to receive data from the server program. The server program should not request any data from the client program yet. This response usually occurs because an application is performing several put operations to the virtual terminal device. After the last put operation by the application, a put/get operation code is usually returned on the read operation.
3	Put/get	Data is returned from this read request and should be sent by the server program to the client program. The operating system is ready to receive data from the server program. The server program should wait for data from the client program.

Figure 35-2 (Page 2 of 2). Read Operation Codes

Value	Response Name	Description
4	Save display	No data is returned from this read request. The operating system expects the server program to obtain the data from the current display and write the data to the virtual terminal. The operating system saves the display for later use, such as returning the display to the server program. The server program must indicate a save-display response on the write operation and send the saved display as data (that is, the saved display must be in the data buffer).
5	Restore display	The data returned is a previously saved display. The server program should send the data to the client program.
6	Read immediate	No data is returned from this read request. The operating system expects the server program to write data to the virtual terminal. Only data from input fields should be written.
8	Read display	No data is returned from this read request. The operating system expects the server program to write data to the virtual terminal. The current display should be written.
A	Cancel invite	No data is returned from this read request. The operating system expects the server program to signal the client program to cancel the outstanding invite operation. When it is canceled, the server program must perform a write operation to the virtual terminal and indicate a cancel invite response. Because the response has no data associated with it, the number of bytes to write must be set to 0 for the write operation.
B	Turn on message light	No data is returned from this read request. A message has been received, and the user should be notified. The server program should signal the client program to turn on a display message indicator light or some other indicator.
C	Turn off message light	No data is returned from this read request. The display message light or some other indicator should be set off.

Error Messages

CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F3 E Data buffer length &1 not valid.
CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.

Send Request for OS/400 Function (QTVSNDRQ) API

Parameters

Required Parameter Group

1	Virtual terminal handle	Input	Char(16)
2	Request	Input	Binary(4)
3	Error code	I/O	Char(*)

The Send Request for OS/400 Function (QTVSNDRQ) API sends a request to the operating system to perform a particular function.

Required Parameter Group

Virtual terminal handle
INPUT; CHAR(16)
The reference code for the open virtual terminal path, created with the Open Virtual Terminal Path (QTVOPNVT) API.

Request
INPUT; BINARY(4)
The request to be processed by the operating system. Valid binary values are:

1 Cancel Previous Request: Allows the server program to cancel the previous OS/400 request. This is similar to selecting option 2 on the AS/400 System Request menu.
2 Send Break Message: Causes the operating system to issue a break message to the virtual terminal. You can use this to determine whether the virtual terminal is still active.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F6 E Request value &1 not valid.
CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.

Write to Virtual Terminal (QTVWRTVT) API

Parameters

Required Parameter Group:

1	Virtual terminal handle	Input	Char(16)
2	Write information	Input	Char(10)
3	Data buffer	Input	Char(*)
4	Number of bytes to write	Input	Binary(4)
5	Error code	I/O	Char(*)

The Write to Virtual Terminal (QTVWRTVT) API writes data from a server program's data buffer to a virtual terminal. You can send one display to the virtual terminal during each write operation. You cannot send partial or multiple displays.

Required Parameter Group

Virtual terminal handle
INPUT; CHAR(16)
The reference code for the open virtual terminal path, created with the Open Virtual Terminal Path (QTVOPNVT) API.

Write information
INPUT; CHAR(10)
Information about the write operation. The information given in each character is as follows:

1 Key flag. Valid values are:
- **0** The Enter key was pressed.
- **1** The System Request key was pressed. The next read operation returns the AS/400 System Request menu.
- **2** The Attention key was pressed. In this case, the number of bytes to write must be 0.
- **3** The Test Request key was pressed.
- **4** The Help-in-Error key was pressed.

2 Operation code. This parameter describes the type of write operation to perform. Valid values and their meanings are:
- **blank** Put/get
- **2** Output only
- **3** Put/get
- **4** Save display
- **A** Cancel invite

For detailed descriptions of these codes, see "Write Operation Codes."

3 Data stream output error. The negative response code, for example X'10030101', is sent as data in the data buffer.
- **blank** 5250 data in data buffer
- **0** 5250 data in data buffer
- **1** Data stream error; SNA response code data is in the data buffer.

4–10 Reserved. These characters must be blank.

Data buffer
INPUT; CHAR(*)
The server program's buffer containing the data to send to the virtual terminal.

The QTVWRTVT API does not lock the data buffer. Thus, other applications should not use the buffer while the API is using it.

Number of bytes to write
INPUT; BINARY(4)
The number of bytes to write. This number must be smaller than or equal to the size of the data buffer. Valid range of numbers is 0 through 24K. This parameter must be 0 if character 1 of the write information parameter is 2.

Some write operations do not write data.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Write Operation Codes

Figure 35-3 describes the operation codes that can be used for the write information parameter.

Value	Name	Description
blank	Put/get	Data is being sent to the virtual terminal. The virtual terminal server program is ready for input.
2	Output only	This write operation is in response to a read request that returned an output-only read operation code.
3	Put/get	See the description above.
4	Save display	This write operation is in response to a read request that returned a save display read operation code. No data is associated with this write operation: thus, the data buffer length must be set to 0.
A	Cancel invite	This write operation is in response to a read request that returned a cancel invite read operation code. No data is associated with this write operation: thus, the data buffer length must be set to 0.

Figure 35-3. Write Operation Codes

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF87D4 E Data sent exceeded the corresponding I/O request.
CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F3 E Data buffer length &1 not valid.
CPF87F4 E Key flag &1 not valid.
CPF87F5 E Operation code response &1 not valid.
CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.

Chapter 36. Virtual Terminal Run-Time Example

To help understand how virtual terminal APIs are used, the following example shows how the OS/400 operating system, server program, client program, and work station device (display and keyboard) interact when processing a system request.

This example starts with the server program waiting for a response from the client program, which is waiting for data from the user (keyboard).

1. System request processing starts when you press the appropriate System Request key on the work station keyboard.
2. The client program informs the server program that the System Request key has been pressed. The protocol used in this case is unique to the particular implementation of these two programs.
3. The server program calls the Write to Virtual Terminal (QTVWRTVT) API for a write request. The flag for the System Request key must be set for this write request. No data is sent to the virtual terminal at this time.
4. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
5. The server program removes the entry from the data queue by calling the Receive Data Queue (QRCVDTAQ) API and then calls the Read from Virtual Terminal (QTVRDVT) API for a read request. The Cancel Invite operation code is returned. No data is received from the virtual terminal at this time.

   To prevent the client program from sending anymore data (screens), the server program informs the client program that it is no longer receiving data from the client program.

   The server program calls the QTVWRTVT API for a write request. The Operation Code parameter is set to Cancel Invite. No data is sent to the virtual terminal at this time.
6. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
7. The server program removes the entry from the data queue by calling the QRCVDTAQ API and then calls the QTVRDVT API for a read request. A Save Screen operation code is returned. No data is received from the virtual terminal at this time.

   The server program gets the current screen. This may require requesting the current screen from the client program.

   The server program calls the QTVWRTVT API for a write request, sending the current screen to the virtual terminal. The Operation Code parameter must be set to Save Screen.
8. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
9. The server program removes the entry from the data queue by calling the QRCVDTAQ API and then calls the QTVRDVT API for a read request. A Put/Get operation code is returned. The data read will be the actual System Request menu.

   The server program sends this data to the client program and waits for a response.
10. The client program updates the display with the System Request menu and waits for a response from the user. The resulting response is sent to the server program.
11. The response is received from the client program, and the server program calls the QTVWRTTV API for a write request, sending the response to the virtual terminal.

    Note: What happens at this point depends on the response to the System Request menu. Additional data may be received from and sent to the virtual terminal. After the response is processed, the following steps occur.
12. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
13. The server program removes the entry from the data queue by calling the QRCVDTAQ API and then performs a call to the QTVRDVT API for a read request. A Put operation code is returned. The data read is the saved (current) screen that was previously written by the server program to the virtual terminal.

    The server program sends the saved screen to the client program but does not wait for a response.
14. The client program updates the work station display with the saved screen.
15. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
16. The server program removes the entry from the data queue by calling the QRCVDTAQ API. An Invite operation code is returned. Note that no data is received from the virtual terminal at this time.
17. The client program is once again waiting for user data, and the server program is waiting for data from the client program.

Part 17. Work Management APIs

Chapter 37. Work Management APIs

This chapter discusses the AS/400 work management APIs that:

- Retrieve information about subsystems and subsystem job queues, and allow you to change storage pool attributes
- Retrieve information from system flight recorders
- Retrieve information about specific jobs
- Retrieve information about job schedule entries

The work management APIs are:

Change Pool Attributes (QUSCHGPA) changes the size and activity level of system storage pools.
Dump Flight Recorder (QWTDMPFR) dumps the contents of the flight recorders for jobs that have them.
Dump Lock Flight Recorder (QWTDMPLF) dumps the contents of the lock flight recorder for the device that is specified in the parameter that is passed to the program.
List Active Subsystems (QWCLASBS) retrieves a list of active subsystems.
List Job (QUSLJOB) lists some or all jobs on the system.
List Job Schedule Entries (QWCLSCDE) lists the entries in the job schedule QDFTJOBSCD.
List Subsystem Job Queues (QWDLSJBQ) lists the job queues for a subsystem.
Retrieve Job Description Information (QWDRJOBD) retrieves information from a job description object.
Retrieve Job Information (QUSRJOBI) retrieves information, such as job attributes and performance data about a specific job.
Retrieve Job Queue Information (QSPRJOBQ) retrieves information associated with a specified job queue.
Retrieve Subsystem Information (QWDRSBSD) retrieves information about a specific subsystem.
Set Lock Flight Recorder (QWTSETLF) turns the lock flight recorder on and off.

This APIs in this chapter are presented in alphabetical order.

Change Pool Attributes (QUSCHGPA) API

Parameters

Required Parameter Group:

1	System pool identifier	Input	Binary(4)
2	New pool size	Input	Binary(4)
3	New pool activity level	Input	Binary(4)

Optional Parameter Group:

4	Message logging	Input	Char(1)
5	Error code	I/O	Char(*)

The Change Pool Attributes (QUSCHGPA) API changes the size and activity level of system storage pools. A system storage pool identifier is returned with the Materialize Resource Management Data (MATRMD) machine interface (MI) instruction. (Note that *system* pool identifiers differ from *subsystem* pool identifiers.) Depending on whether the base pool, shared pool, or private subsystem pool is to be changed, the QUSCHGPA API determines the appropriate command to use and then issues that command. This is similar to the function provided on the System Status display, where you can change the system storage pool size and activity level interactively.

You can use the QUSCHGPA API to tune storage pools (without having to know which subsystem monitor allocated the pool and without having to determine whether or not a pool is a shared storage pool). The Work with System Status (WRKSYSSTS) and the Work with Subsystems (WRKSBS) commands combined provide similar functions.

Authorities and Locks

Subsystem Description Authority
*OBJOPR
Description Library Authority
*OBJOPR

Required Parameter Group

System pool identifier
INPUT; BINARY(4)
This identifies which pool is to be changed. This number corresponds to the number returned on option nine of the MATRMD MI instruction using the System C/400 PRPQ. This also corresponds to the identifier shown on the Work with System Status display. This parameter is a value ranging from 1 to 16 where pool 1 is the machine pool, and pool 2 is the base pool.

New pool size
INPUT; BINARY(4)
The size of the pool in kilobytes, where one kilobyte is 1024 bytes. If you do not want the pool size to be changed, you must specify a value of -1 for this parameter. The minimum value is 32 kilobytes.

New pool activity level
INPUT; BINARY(4)
The activity level for the pool. If you do not want the activity level to be changed, you must specify a value of -1 for this parameter. You cannot change the activity level of the machine pool.

Optional Parameter Group

Message logging
INPUT; CHAR(1)
Whether messages reporting that a change was made are written to the current job's job log and to the QHST message log. This affects the logging of

change-related messages only; it does not affect the logging of error messages. Valid values are:

Y Log change messages.
N Do not log change messages.

If this parameter is omitted, Y is used and change messages are logged.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

The following table summarizes the values you can specify for the system pool identifier, the new pool size, and the new pool activity level.

System Pool Identifier	New Pool Size	New Pool Activity Level
1 Machine pool	-1 or ≥ 256	-1
2 Base pool	-1 or ≥ 32	-1 or 1 through 32 767
3 to 16	≥ 1	1 through 32 767

For pools 3-16, both size and pool activity level must be specified.

In some cases, pool size changes do not take effect immediately. For example, a save or restore operation might be using some of the storage allocated to a pool, or the system might be using some of the storage allocated to the base pool. The size is changed only when the storage being used is free again.

The base pool holds all unused main storage on the system that is not allocated to other shared or private pools. As subsystems are started and allocate storage for their shared and private storage pools, that storage comes from the base pool. The base pool (pool number 2) size is what is left after pool 1 and pools 3 through 16 are subtracted from the total main storage. The QBASPOOL system value is a minimum size, and it is not the actual size of the base pool. At this minimum size, the system does not allow additional storage requests. For this reason, you must calculate the storage requirements for all pools on the system, including the base pool, and then run this API.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CA0 E System pool &1 does not exist.
CPF3CA1 E Pool size &1 is not valid.
CPF3CA2 E Activity level &1 is not valid.
CPF3CA3 E Pool &1 is not in use.
CPF3CA4 E Changing machine pool activity level is not allowed.
CPF3CA5 E Both pool size and activity level are required.
CPF3CA6 E Message logging value &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.

Example: Changing System Storage Pool Attributes

The following is an example of how to change system storage pool attributes using the QUSCHGPA API:

```
Pseudocode
     .
     .
     .
MATRMD (OPERAND1, OPERAND2);
DO             /* Do loop for each pool in use    */
     .
     .         /* Calculate the desired pool size
                  and activity level              */
     .
END
DO             /* Do loop for each pool in use    */
CALL QUSCHGPA (POOLID, POOLSIZE, POOLACTLVL);
               /* Change pool attributes          */
END
     .
     .
     .
```

Dump Flight Recorder (QWTDMPFR) API

The Dump Flight Recorder (QWTDMPFR) API dumps the contents of flight recorders for jobs that have them. A **flight recorder** is an object that stores trace information to record a history of what has happened in system programs. The flight recorder contains only information that helps to identify the flow of system programs and status information.

The following types of jobs have flight recorders:

- Subsystem monitors
- System jobs

The QWTDMPFR API has no parameters.

You can use the QWTDMPFR API to collect information for your IBM service representative. This API dumps the contents of job flight recorders to a spooled file. You can then collect the files and submit them to your IBM service representative for debugging.

Dump Lock Flight Recorder (QWTDMPLF) API

Parameters

Required Parameter:

1	Device name	Input	Char(10)

Optional Parameter:

2	Error code	I/O	Char(*)

The Dump Lock Flight Recorder (QWTDMPLF) API dumps the following information into spooled files:

- The contents of the lock flight recorder for the device specified in the parameter passed to the program
- QSYSARB job log
- QLUS job log
- Job logs of the active jobs that have used the device as indicated in the lock flight recorder data
- The history log (QHST)
- Device description of the device
- Controller description of the controller to which the device is attached
- Line description of the line to which the controller is attached
- A Work with Object Locks (WRKOBJLCK) listing for the device
- A Work with Configuration Status (WRKCFGSTS) listing for the controller
- The subsystem description of active subsystems that have touched the device
- Associated internal system objects

You can use the QWTDMPLF API to collect information for your IBM service representative.

Required Parameter

Device name
INPUT; CHAR(10)
The name of the device for which flight recorder information will be dumped.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF119C E Value &1 specified for parameter is not valid.
CPF3CF1 E Error code parameter not valid.
CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.

List Active Subsystems (QWCLASBS) API

Parameters

Required Parameter Group:

1	Receiving user space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	Error code	I/O	Char(*)

The List Active Subsystems (QWCLASBS) API retrieves a list of active subsystems. QWCLASBS replaces any subsystem list that already exists in the user space. It does not add the new list to an existing one. To retrieve more information about active subsystems, see "Retrieve Subsystem Information (QWDRSBSD) API" on page 37-27.

Authorities and Locks

User Space Authority *CHANGE
Library Authority *USE
User Space Lock *EXCLRD

Required Parameter Group

Receiving user space and library name
INPUT; CHAR(20)
The user space that receives the list, and the library in which it is located. The first 10 characters contain the user space name. The second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The format to use for the list of active subsystems. You can use this format name:

SBSL0100 Basic subsystem list. See "Format of the Generated List."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated List

The list of active subsystems that the QWCLASBS API returns into the user space consists of:

- A user area
- A generic header
- An input parameter section
- A list data section

The user area and generic header are described in "User Space Format for List APIs" on page 2-7. The remaining items are described in the following sections. For detailed descriptions of the fields in the tables, see "Field Descriptions" on page 37-4.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library specified

SBSL0100 Format: This section is repeated for each active subsystem.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Subsystem description name
10	A	CHAR(10)	Subsystem description library name

Field Descriptions

Subsystem description library name. The name of the library in which the active subsystem description resides.

Subsystem description name. The name of the active subsystem about which information is being returned.

User space library specified. The library name or special value specified in the call to this API.

User space name. The name of the user space that receives the list.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF8122 E &8 damage on library &4.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.

List Job (QUSLJOB) API

Parameters

Required Parameter Group:

1	Receiving user space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified job name	Input	Char(26)
4	Status	Input	Char(10)

Optional Parameter:

5	Error code	I/O	Char(*)

The List Job (QUSLJOB) API generates a list of all or some jobs on the system. The generated list replaces any existing list in the user space.

The QUSLJOB API produces a list similar to the list produced by the Work with User Job (WRKUSRJOB) command. However, you cannot use the API and the command interchangeably. Because they perform different authority checks, they might produce different lists. You might be able to list certain objects with one method but not with the other method.

Authorities and Locks

User Space Authority *CHANGE
Library Authority *USE
User Space Lock *EXCLRD

Required Parameter Group

Receiving user space and library name
INPUT; CHAR(20)
The user space that is to receive the generated list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The format of the job list to be returned. You must use the following format name:

JOBL0100 Basic job list. For more information, see "Format of the Generated List" on page 37-5.

Qualified job name
INPUT; CHAR(26)
The name of the job to be included in the list. The qualified job name has three parts:

Job name
CHAR(10). A specific job name, a generic name, or one of the following special values:
* Only the job that this program is running in. The rest of the qualified job name parameter must be blank.
***CURRENT** All jobs with the current job's name.
***ALL** All jobs. The rest of the job name parameter must be specified.

User name
CHAR(10). A specific user profile name, a generic name, or one of the following special values:
***CURRENT** Jobs with the current job's user profile.
***ALL** Jobs with the specified job name, regardless of the user name. The rest of the job name parameter must be specified.

Job number
CHAR(6). A specific job number or the following special value:

***ALL** Jobs with the specified job name and user name, regardless of the job number. The rest of the job name parameter must be specified.

If only the job name and no other qualifying values are given, the system searches all the jobs on the system for the job name. If the system finds duplicates of the specified job, it displays a list of messages containing the qualified job names of all duplicates.

Status
INPUT; CHAR(10)
The status of the jobs to be included in the list. The special values supported are:

***ACTIVE** Active jobs. This includes group jobs, system request jobs, and disconnected jobs.
***JOBQ** Jobs currently on job queues.
***OUTQ** Jobs that have completed running but still have output on an output queue.
***ALL** All jobs, regardless of status.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated List

The job list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

For details about the user area and generic header, see "User Space Format for List APIs" on page 2-7. For details about the remaining items, see the following sections. For detailed descriptions of the fields in the list returned, see "Field Descriptions."

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job name specified
10	A	CHAR(10)	User name specified
20	14	CHAR(6)	Job number specified
26	1A	CHAR(10)	Status

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job name used
10	A	CHAR(10)	User name used
20	14	CHAR(6)	Job number used

JOBL0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job name used
10	A	CHAR(10)	User name used
20	14	CHAR(6)	Job number used
26	1A	CHAR(16)	Internal job identifier
42	2A	CHAR(10)	Status

When you retrieve list entry information from a user space, you should use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Appendix A, "Examples."

Field Descriptions

Internal job identifier. A value sent to other APIs to speed the process of locating the job on the system. Only APIs described in this manual use this identifier. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Job name specified. The name of the job as specified in the call to the API.

Job name used. The name of the job as identified to the system. For an interactive job, the system assigns the job the name of the work station where the job started; for a batch job, you specify the name in the command when you submit the job.

Job number specified. The job number as specified in the call to the API.

Job number used. The system-assigned job number.

Status. The status of the job. The valid values are:

**ACTIVE* The job has started, and it can use system resources (processing unit, main storage, and so on). This does not guarantee that the job is currently running, however. For example, an active job may be in one of the following states where it is not in a position to use system resources:

- The Hold Job (HLDJOB) command holds the job; the Release the (RLSJOB) command allows the job to run again.
- The Transfer Group Job (TFRGRPJOB) or Transfer Secondary Job (TFRSECJOB) command suspends the job. When control returns to the job, the job can run again.
- The job is disconnected using the Disconnect Job (DSCJOB) command. When the interactive user signs back on, thereby connecting back into the job, the job can run again.
- The job is waiting for any reason. For example, with an inquiry message when the job receives the reply, the job can start running again.

*JOBQ The job is currently on a job queue. The job possibly was previously active and was placed back on the job queue because of the Transfer Job (TFRJOB) or Transfer Batch Job (TFRBCHJOB) command, or the job was never active because it was just submitted.

*OUTQ The job has completed running and has spooled output that has not yet printed.

User name specified. The user name as specified in the call to the API.

User name used. The user profile under which the job is run. The user name is the same as the user profile name and can come from several different sources depending on the type of job.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CB1 E Value &1 for job status is not valid.
CPF3CB2 E Value specified for job parameter is not valid.
CPF3CF1 E Error code parameter not valid.
CPF3C20 E Error found by program &1.
  CPD3C21 D Format name &1 is not valid.
  CPD3CB1 D Value &1 for job status is not valid.
  CPD3CB2 D Value specified for job parameter is not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.

List Job Schedule Entries (QWCLSCDE) API

Parameters

Required Parameter Group:

1	User space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	Job schedule entry name	Input	Char(10)
4	Continuation handle	Input	Char(16)
5	Error code	I/O	Char(*)

The List Job Schedule Entries (QWCLSCDE) API lists the entries in the job schedule, QDFTJOBSCD. A subset of the list can be created by using the job schedule entry name parameter. The generated list replaces any existing list in the user space.

The QWCLSCDE API produces a list similar to the list produced by the Work with Job Schedule Entries (WRKJOBSCDE) command.

Authorities and Locks

User Space Authority *CHANGE
User Space Library Authority *USE
User Space Lock *EXCLRD
Job Schedule Entry Authority *USE if using format SCDL0100; *JOBCTL or the adder of the entry if using format SCDL0200
Job Schedule Authority *USE
Job Schedule Library Authority *READ
Job Schedule Lock *SHRRD

Required Parameter Group

User space and library name
INPUT; CHAR(20)
The user space that is to receive the created list. The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The content and format of the information returned for each member. The possible format names are:

SCDL0100 Basic job schedule entries list.
SCDL0200 Detailed job schedule entries list. This format requires more processing than the SCDL0100 format.

For more information, see "SCDL0100 Format" on page 37-7 or "SCDL0200 Format" on page 37-7.

Job schedule entry name
INPUT; CHAR(10)
The job schedule entry about which to retrieve information. This can be used to create a subset of job schedule entries by using the following values:

***ALL**	All of the job schedule entries are returned in the list.
generic*	All of the job schedule entries beginning with the generic value are returned in the list.
name	The job schedule entries with the given name are returned in the list.

Continuation handle
INPUT; CHAR(16)
The value returned to the user in the header section when a partial list is returned. The possible values are:

blank	This will start at the beginning of the list.
value	The entries after this value matching the job schedule entry name specified will be returned in the list. For more information on using this value to process a partial list, see "Partial List Considerations" on page 2-11.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated Lists

The file member list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section:
  - SCDL0100 format
  - SCDL0200 format

When you retrieve list entry information from a user space, you must use the entry size returned in the generic header. The size of each entry may be padded at the end. If you do not use the entry size, the result may not be valid. For examples of how to process lists, see Appendix A, "Examples."

For details about the user area and generic header, see "User Space Format for List APIs" on page 2-7. For details about the remaining items, see the following sections. For detailed descriptions of the fields in the list returned, see "Field Descriptions" on page 37-8.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	Job schedule entry name
44	2C	CHAR(16)	Continuation handle

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job schedule entry name used
10	A	CHAR(16)	Continuation handle

SCDL0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Information status
1	1	CHAR(10)	Job name
11	B	CHAR(10)	Entry number
21	15	CHAR(10)	Scheduled date
31	1F	CHAR(70)	Scheduled days
101	65	CHAR(6)	Scheduled time
107	6B	CHAR(10)	Frequency
117	75	ARRAY(5) of CHAR(10)	Relative day of the month
167	A7	CHAR(10)	Recovery action
177	B1	CHAR(10)	Next submission date
187	BB	CHAR(10)	Status
197	C5	CHAR(10)	Job queue name
207	CF	CHAR(10)	Job queue library name
217	D9	CHAR(10)	User profile of entry adder
227	E3	CHAR(10)	Last submission date
237	ED	CHAR(6)	Last submission time
243	F3	CHAR(50)	Text
293	125	CHAR(23)	Reserved

SCDL0200 Format

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format SCDL0100
316	13C	CHAR(10)	Job queue status
326	146	ARRAY(20) of CHAR(10)	Dates omitted
526	20E	CHAR(10)	Job description name

Offset		Type	Field
Dec	Hex		
536	218	CHAR(10)	Job description library name
546	222	CHAR(10)	User profile for submitted job
556	22C	CHAR(10)	Message queue name
566	236	CHAR(10)	Message queue library name
576	240	CHAR(10)	Save entry
586	24A	CHAR(10)	Last submission job name
596	254	CHAR(10)	Last submission user name
606	25E	CHAR(6)	Last submission job number
612	26E	CHAR(10)	Last attempted submission date
622	26E	CHAR(6)	Last attempted submission time
628	274	CHAR(10)	Status of last attempted submission
638	27E	CHAR(2)	Reserved
640	280	BINARY(4)	Length of command string
644	284	CHAR(512)	Command

Field Descriptions

Command. A command that runs in the submitted batch job if the routing program used when this batch job is started is the IBM-supplied default routing program QCMD. Because this command is used for the request data, this parameter takes the place of any value specified for the RQSDTA parameter in the job description.

Continuation handle. The value used to process a partial list. This value is put in the header section when a partial list is returned. For more information on processing a partial list, see "Partial List Considerations" on page 2-11.

Dates omitted. Specifies up to 20 dates when the job should not be submitted. The dates will be in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century flag indicates the twentieth century, and a 1 indicates the twenty-first century. If no omit dates are specified, this field contains hexadecimal zeros.

Entry number. The entry number assigned to the job schedule entry. The entry number can range from 000001 through 999999.

Format name. The content and format of the information returned for each job schedule entry.

Frequency. How often the job schedule entry is to be submitted to run. Valid values are:

**ONCE*	The job schedule entry does not repeat.
**WEEKLY*	The job schedule entry is submitted on the same day of each week at the scheduled time.
**MONTHLY*	The job schedule entry is submitted on the same day of each month at the scheduled time.

Information status. Whether or not the entry information could be successfully retrieved.

blank	No errors occurred. All information is returned for this entry.
A	Insufficient authority to entry. Only the SCDL0100 information is returned for this entry. The rest of the entry is blank.
L	The entry is locked. Only the SCDL0100 information is returned for this entry. The rest of the entry is blank.

Job description name. The name of the job description used for the job schedule entry.

**USRPRF*
: The job description in the user profile under which the job runs is used as the job description of the job schedule entry.

job description name
: The name of the job description used for the job schedule entry.

Job description library name. The name of the library in which the job description is located.

Job name. The job name associated with the job schedule entry.

Job queue library name. The name of the library in which the specified job queue resides.

Job queue name. The name of the job queue where the job should be placed when it is submitted. Valid values are:

**JOBD*	The job is placed in the job queue named in the specified job description.
job queue name	The name of the job queue where the job is placed when it is submitted.

Job queue status. The current status of the job queue. Valid values are:

blank	The job queue name specified *JOBD, the job queue could not be found, or the job queue is damaged.
HLD	The job queue is held. The job queue is not allocated to a subsystem.
RLS	The job queue is released. The job queue is not allocated to a subsystem.
HLD/SBS	The job queue is held. The job queue is allocated to a subsystem.
RLS/SBS	The job queue is released. The job queue is allocated to a subsystem.
LOCKED	The job queue is locked, and the status could not be obtained.

Job schedule entry name used. The job name used to identify the job schedule entry.

Last attempted submission date. The date on which a job could have been last submitted. The value is returned in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century indicates the twentieth century, and a 1 indicates the twenty-first century. If no submission has been attempted, this field contains hexadecimal zeros.

Last attempted submission time. The time at which a job could have been last submitted from this entry. The value is returned in the format HHMMSS, where HH is hours, MM is minutes, and SS is seconds. If no submission has been attempted, this field contains hexadecimal zeros.

Last submission date. The date on which a job was last submitted from this entry. The value is returned in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century indicates the twentieth century, and a 1 indicates the twenty-first century. If there has been no previous submission, this field contains hexadecimal zeros.

Last submission job name. The job name of the last job that was submitted from this entry. If there has been no previous submission, this field contains blanks.

Last submission job number. The job number of the last job that was submitted from this entry. If there has been no previous submission, this field contains blanks.

Last submission time. The time at which a job was last submitted from this entry. The value is returned in the format HHMMSS, where HH is hours, MM is minutes, and SS is seconds. If there has been no previous submission, this field contains hexadecimal zeros.

Last submission user name. The user name of the last job that was submitted from this entry. If there has been no previous submission, this field contains blanks.

Length of command string. The length of the command string specified in the command field.

Message queue name. The name of the message queue, if any, to which a completion message is sent when the job is submitted. A completion message is sent when the submitted job has completed running, and error messages are sent if the Submit Job (SBMJOB) command fails for some reason. Valid values are:

**USRPRF* A completion message is sent to the message queue specified in the user profile associated with the job schedule entry.

**NONE* Messages are not sent to a user-specified message queue. In this case, completion messages are not sent, but error messages are sent to the QSYSOPR message queue.

message queue name
The name of the message queue where the messages are sent.

Message queue library name. The library in which the message queue is located.

Next submission date. The next date that a job from this entry is scheduled to be submitted. The next submission date is returned in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century indicates the twentieth century, and a 1 indicates the twenty-first century. If this entry has a status of SAV, this field contains hexadecimal zeros.

Recovery action. The action that will happen if the job cannot be submitted at the designated time because the system is powered down or the system is in the restricted state. The action specified by this parameter then occurs when the system is IPLed or when the system comes out of the restricted state. This parameter does not pertain to the situation where a job was held (by the user) when the designated time elapsed and then released (by the user) at a later time. Also, the recovery action does not pertain to timer events that elapse as a result of changes to the QTIME system value. Valid values are:

**SBMRLS* Submit the job in the released state (RLS).
**SBMHLD* Submit the job in the held state (HLD).
**NOSBM* No job is submitted.

Relative day of the month. The relative day of the month the job should be submitted to run. A total of five values can be returned. If no relative day of the month was specified, this area contains blanks. Valid values are:

1-5 The job should be submitted on the specified day of the week every first, second, third, fourth, or fifth week of the month.
**LAST* The job should be submitted on the last specified day of the week each month.

Reserved. An ignored field.

Save entry. Whether or not an entry that has FRQ(*ONCE) specified should be kept in the job schedule after the job has been submitted.

**NO* This entry will not be kept after the job is submitted.
**YES* This entry will be kept after the job is submitted.

Scheduled date. The date that the job will be submitted. Valid values are:

**CURRENT*
The current date will be used.
**MONTHSTR*
The first day of the month will be used.
**MONTHEND*
The last day of the month will be used.
**NONE* A scheduled date was not specified.
date An actual date in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century indicates the twentieth century, and a 1 indicates the twenty-first century.

Scheduled days. The day of the week that the job will be submitted. A total of seven values can be returned. Valid values are:

**ALL* The job will be submitted every day. This cannot be specified with any other values.
**NONE* A scheduled day is not specified. This cannot be specified with any other values.
**MON* The job will be submitted on Monday.
**TUE* The job will be submitted on Tuesday.
**WED* The job will be submitted on Wednesday.
**THU* The job will be submitted on Thursday.
**FRI* The job will be submitted on Friday.
**SAT* The job will be submitted on Saturday.
**SUN* The job will be submitted on Sunday.

Scheduled time. The time (on the scheduled date) when the job will be submitted to run. This value is returned in the format HHMMSS, where HH is the hours, MM is the minutes, and SS is the seconds.

Status. The status of the job schedule entry. Valid values are:

SCD The entry is scheduled.
HLD The entry is held.
SAV The entry is saved.

Status of last attempted submission. The action that occurred the last time the system could have submitted a job from this entry. Valid values are:

0 Job not previously submitted.
1 Job successfully submitted.
2 Last job submission failed. Check the message queue for details.
3 Job not submitted due to held status.
4 Job submitted after scheduled time as specified by recovery action.
5 Job not submitted as specified by recovery action.

Text. Text that briefly describes the job schedule entry. If no text was specified, this field contains blanks.

User profile for submitted job. The user profile under which the job will be submitted. Valid values are:

**JOBD* The user profile named in the specified job description is used for the job.
user name The name of the user profile that is used for the job.

User profile of entry adder. The user profile that created this entry.

User space library name. The library name or special value specified in the call to this API.

User space name. The name of the user space that receives the list.

Error Messages

CPF1629 Not authorized to job schedule &1.
CPF1632 Job schedule entry &3 number &4 damaged.
CPF1637 Job schedule &1 in library &2 in use.
CPF1640 Job schedule &1 in library &2 does not exist.
CPF1641 Job schedule &1 in library &2 damaged.
CPF1643 Job schedule entry name not valid.
CPF3C21 Format name &1 is not valid.
CPF3CF1 Error code parameter not valid.
CPF811A User space &4 in &9 damaged.
CPF8122 &8 damage on library &4.
CPF812C Job schedule &4 in &9 damaged.
CPF9801 Object &2 in library &3 not found.
CPF9802 Not authorized to object &2 in &3.
CPF9803 Cannot allocate object &2 in library &3.
CPF9807 One or more libraries in library list deleted.
CPF9808 Cannot allocate one or more libraries on library list.
CPF9810 Library &1 not found
CPF9820 Not authorized to use library &1.
CPF9830 Cannot assign library &1.
CPF9838 User profile storage limit exceeded.

List Subsystem Job Queues (QWDLSJBQ) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	List format	Input	Char(8)
3	Subsystem and library name	Input	Char(20)
4	Error code	I/O	Char(*)

The List Subsystem Job Queues (QWDLSJBQ) API lists the job queues for a subsystem. It also gives the job queue allocation status, indicating whether the specified subsystem is active and has allocated this job queue or not. QWDLSJBQ replaces any job queue list that already exists in the user space.

Authorities and Locks

User Space Authority *CHANGE
User Space Library Authority *USE
User Space Lock *EXCLRD
Subsystem Description Authority *USE
Subsystem Description Library Authority *READ

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The user space that receives the list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

List format
INPUT; CHAR(8)
The format to use for the list of job queues. You can use this format name:

SJQL0100 Basic job queue list. For details, see "Format of the Generated List."

Subsystem and library name
INPUT; CHAR(20)
The subsystem about which to retrieve information, and the library in which the subsystem description is located. The first 10 characters contain the subsystem name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated List

The list of job queues that the QWDLSJBQ API returns into the user space consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

The user area and generic header are described in "User Space Format for List APIs" on page 2-7. The remaining items are described in the following sections. For detailed descriptions of the fields in the tables, see "Field Descriptions."

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name specified
20	14	CHAR(10)	Subsystem name
30	1E	CHAR(10)	Subsystem library name

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Subsystem name
10	A	CHAR(10)	Subsystem library name used

SJQL0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job queue name
10	A	CHAR(10)	Job queue library name
20	14	BINARY(14)	Sequence number
24	18	CHAR(10)	Allocation indicator

Field Descriptions

Allocation indicator. A value indicating whether or not the job queue is allocated to the specified subsystem. Valid values are:

**NO* The subsystem has not allocated this job queue. Either this subsystem is inactive, or another subsystem has allocated the job queue.
**YES* The subsystem is active and has allocated this job queue.

Job queue library name. The name of the library in which the specified job queue resides.

Job queue name. The name of a job queue specified in a subsystem description job queue entry.

Sequence number. The job queue entry sequence number. The subsystem uses this number to determine the order in which job queues are processed. Jobs from the queue with the lowest sequence number are processed first.

Subsystem library name specified. The name or special value specified in the call to this API for the library in which the subsystem description resides.

Subsystem library name used. The name of the library in which the subsystem description resides.

Subsystem name. The name of the active subsystem about which information is being returned.

User space library name specified. The library name or special value specified in the call to this API.

User space name. The name of the user space that receives the list.

Error Messages

CPF1605 E Cannot allocate subsystem description &1.
CPF1606 E Error during allocation of subsystem &1.
CPF1607 E Previous request pending for subsystem &1.
CPF1608 E Subsystem description &1 not found.

CPF1619 E	Subsystem description &1 in library &2 damaged.
CPF1835 E	Not authorized to subsystem description.
CPF3CF1 E	Error code parameter not valid.
CPF3C21 E	Format name &1 is not valid.
CPF811A E	User space &4 in &9 damaged.
CPF8122 E	&8 damage on library &4.
CPF9801 E	Object &2 in library &3 not found.
CPF9802 E	Not authorized to object &2.
CPF9803 E	Cannot allocate object &2 in library &3.
CPF9807 E	One or more libraries in library list deleted.
CPF9808 E	Cannot allocate one or more libraries on library list.
CPF9810 E	Library &1 not found.
CPF9820 E	Not authorized to use library &1.
CPF9830 E	Cannot assign library &1.
CPF9838 E	User profile storage limit exceeded.

Retrieve Job Description Information (QWDRJOBD) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified job description name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Job Description Information (QWDRJOBD) API retrieves information from a job description object and places it into a single variable in the calling program. The amount of information returned depends on the size of the variable. The information returned is the same information returned by the Display Job Description (DSPJOBD) command.

Authorities and Locks

Job Description Object Authority *USE
Library Authority *USE

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. You can specify the size of this area to be smaller than the format requested if you specify the length of receiver variable parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If this value is larger than the actual size of the receiver variable, the result may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the job description information to be returned. You can use this format:

JOBD0100 Basic job description information. For details, see "JOBD0100 Format."

Qualified job description name
INPUT; CHAR(20)
The name of the job description whose contents are to be retrieved. The first 10 characters contain the name of the job description, and the second 10 characters contain the name of the library where the job description is located. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

JOBD0100 Format

The following table describes the information that is returned in the receiver variable for the JOBD0100 format. For detailed descriptions of the fields, see "Field Descriptions" on page 37-13.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Job description name
18	12	CHAR(10)	Job description library name
28	1C	CHAR(10)	User name
38	26	CHAR(8)	Job date
46	2E	CHAR(8)	Job switches
54	36	CHAR(10)	Job queue name
64	40	CHAR(10)	Job queue library name
74	4A	CHAR(2)	Job queue priority
76	4C	CHAR(10)	Hold on job queue
86	56	CHAR(10)	Output queue name
96	60	CHAR(10)	Output queue library name
106	6A	CHAR(2)	Output queue priority
108	6C	CHAR(10)	Printer device name
118	76	CHAR(30)	Print text
148	94	BINARY(4)	Syntax check severity
152	98	BINARY(4)	End severity

Offset		Type	Field
Dec	Hex		
156	9C	BINARY(4)	Message logging severity
160	A0	CHAR(1)	Message logging level
161	A1	CHAR(10)	Message logging text
171	AB	CHAR(10)	Logging of CL programs
181	B5	CHAR(10)	Inquiry message reply
191	BF	CHAR(13)	Device recovery action
204	CC	CHAR(10)	Time slice end pool
214	D6	CHAR(15)	Accounting code
229	E5	CHAR(80)	Routing data
309	135	CHAR(50)	Text description
359	167	CHAR(1)	Reserved
360	168	BINARY(4)	Offset to initial library list
364	16C	BINARY(4)	Number of libraries in initial library list
368	170	BINARY(4)	Offset to request data
372	174	BINARY(4)	Length of request data
376	174	CHAR(*)	Reserved
*	*	ARRAY (*) of CHAR(11)	Initial library list
*	*	CHAR(*)	Request data

Field Descriptions

Accounting code. An identifier assigned to jobs that use this job description. This code is used to collect system resource use information. If the special value *USRPRF is specified, the accounting code used for jobs using this job description is obtained from the job's user profile.

Bytes available. The length of all data available to return. All available data is returned if enough space is provided.

Bytes returned. The length of all data actually returned. If the data is truncated because the receiver variable was not sufficiently large to hold all of the data available, this value will be less than the bytes available.

Device recovery action. The action to take when an I/O error occurs for the interactive job's requesting program device. The possible values are:

**SYSVAL*
: The value in the system value QDEVRCYACN at the time the job is started is used as the device recovery action for this job description.

**MSG*
: Signals the I/O error message to the application and lets the application program perform error recovery.

**DSCMSG*
: Disconnects the job when an I/O error occurs. When the job reconnects, the system sends a message to the application program, indicating the job has reconnected and that the work station device has recovered.

**DSCENDRQS*
: Disconnects the job when an I/O error occurs. When the job reconnects, the system sends the End Request (ENDRQS) command to return control to the previous request level.

**ENDJOB*
: Ends the job when an I/O error occurs. A message is sent to the job's log and to the history log (QHST) indicating the job ended because of a device error.

**ENDJOBNOLIST*
: Ends the job when an I/O error occurs. There is no job log produced for the job. The system sends a message to the history log (QHST) indicating the job ended because of a device error.

End severity. The message severity level of escape messages that can cause a batch job to end. The batch job ends when a request in the batch input stream sends an escape message, whose severity is equal to or greater than this value, to the request processing program. The possible values are from 0 through 99.

Hold on job queue. Whether jobs using this job description are put on the job queue in the hold condition. The possible values are *YES and *NO.

Initial library list. The initial library list that is used for jobs that use this job description. Only the libraries in the user portion of the library list are included.

Note: The data is an array of 11-byte entries, each entry consisting of a 10-byte library name left-justified with a blank pad at the end. The number of libraries in the initial library list tells how many entries are contained in the array.

Inquiry message reply. How inquiry messages are answered for jobs that use this job description.

**RQD*
: The job requires an answer for any inquiry messages that occur while the job is running.

**DFT*
: The system uses the default message reply to answer any inquiry messages issued while the job is running. The default reply is either defined in the message description or is the default system reply.

**SYSRPYL*
: The system reply list is checked to see if there is an entry for an inquiry message issued while the job is running. If a match occurs, the system uses the reply value for that entry. If no entry exists for that message, the system uses an inquiry message.

Job date. The date that will be assigned to jobs using this job description when they are started. The possible values are:

**SYSVAL*
The value in the QDATE system value is used at the time the job is started.
job-date
The date to be used at the time the job is started. This date is in the format specified for the DATFMT job attribute.

Job description library name. The name of the library in which the job description resides.

Job description name. The name of the job description about which information is being returned.

Job queue library name. The library of the job queue into which batch jobs using this job description are placed.

Job queue name. The name of the job queue into which batch jobs using this job description are placed.

Job queue priority. The scheduling priority of each job that uses this job description. The highest priority is 1 and the lowest priority is 9.

Job switches. The initial settings for a group of eight job switches used by jobs that use this job description. These switches can be set or tested in a program and used to control a program's flow. The possible values are '0' (off) and '1' (on).

Length of request data. The length of all available request data, in bytes. If the receiver variable was not sufficiently large to hold all of the request data available, the amount of request data actually returned may be less than this value.

Logging of CL programs. Whether or not messages are logged for CL programs that are run. The possible values are *YES and *NO.

Message logging level. The type of information logged. Possible types are:

0 No messages are logged.
1 All messages sent to the job's external message queue with a severity greater than or equal to the message logging severity are logged.
2 The following information is logged:
- Level 1 information.
- Requests or commands from CL programs for which the system issues messages with a severity code greater than or equal to the logging severity.
- All messages associated with those requests or commands that have a severity code greater than or equal to the logging severity.

3 The following information is logged:
- Level 1 information.
- All requests or commands from CL programs.
- All messages associated with those requests or commands that have a severity greater than or equal to the logging severity.

4 The following information is logged:
- All requests or commands from CL programs.
- All messages with a severity code greater than or equal to the logging severity.

Message logging severity. The minimum severity level that causes error messages to be logged in the job log. The possible values are from 0 through 99.

Message logging text. The level of message text that is written in the job log or displayed to the user when an error message is created according to the logging level and logging severity. The possible values are:

**MSG* Only the message is written to the job log.
**SECLVL*
Both the message and the message help for the error message are written to the job log.
**NOLIST*
If the job ends normally, there is no job log. If the job ends abnormally (the job end code is 20 or higher), there is a job log. The messages appearing in the job's log contain both the message and the message help.

Number of libraries in initial library list. The number of libraries in the user portion of the initial library list. Up to 25 libraries can specified.

Offset to initial library list. The offset from the beginning of the structure to the start of the initial library list.

Offset to request data. The offset from the beginning of the structure to the start of the request data.

Output queue library name. The name of the library in which the output queue resides.

Output queue name. The name of the default output queue that is used for spooled output produced by jobs that use this job description.

**USRPRF*
The output queue name for jobs using this job description is obtained from the user profile of the job at the time the job is started.
**DEV*
The output queue with the same name as the printer device for this job description is used.
**WRKSTN*
The output queue name is obtained from the device description from which this job is started.
output-queue-name
The name of the output queue for this job description.

Output queue priority. The output priority for spooled files that are produced by jobs using this job description. The highest priority is 1, and the lowest priority is 9.

Print text. The line of text (if any) that is printed at the bottom of each page of printed output for jobs using this job description. If the special value *SYSVAL is specified, the value in the system value QPRTTXT is used for jobs using this job description.

Printer device name. The name of the printer device or the source for the name of the printer device that is used for all spooled files created by jobs that use this job description.

**USRPRF*
: The printer device name is obtained from the user profile of the job at the time the job is started.

**SYSVAL*
: The value in the system value QPRTDEV at the time the job is started is used as the printer device name.

**WRKSTN*
: The printer device name is obtained from the work station where the job was started.

printer-device-name
: The name of the printer device that is used with this job description.

Request data. The request data that is placed as the last entry in the job's message queue for jobs that use this job description. The possible values are:

**NONE* No request data is placed in the job's message queue.

**RTGDTA*
: The data specified in the routing data parameter is placed as the last entry in the job's message queue.

request-data
: The request data to use for jobs that use this job description.

Reserved. An ignored field.

Routing data. The routing data that is used with this job description to start jobs. The possible values are:

QCMDI The default routing data QCMDI is used by the IBM-supplied interactive subsystem to route the job to the IBM-supplied control language processor QCMD in the QSYS library.

**RQSDTA*
: Up to the first 80 characters of the request data specified in the request data field are used as the routing data for the job.

routing-data
: The routing data to use for jobs that use this job description.

Syntax check severity. Whether requests placed on the job's message queue are checked for syntax as CL commands, and the message severity that causes a syntax error to end processing of a job. The possible values are:

-1 The request data is not checked for syntax as CL commands. This is equivalent to *NOCHK.

0 – 99 Specifies the lowest message severity that causes a running job to end. The request data is checked for syntax as CL commands, and, if a syntax error occurs that is greater than or equal to the error message severity specified here, the running of the job that contains the erroneous command is suppressed.

Text description. The user text, if any, used to briefly describe the job description.

Time slice end pool. Whether interactive jobs using this job description should be moved to another main storage pool when they reach time slice end. The possible values are:

**SYSVAL* The system value is used.

**NONE* The job is not moved when it reaches time slice end.

**BASE* The job is moved to the base pool when it reaches time slice end.

User name. The name of the user profile associated with this job description. If *RQD is specified, a user name is required to use the job description.

Error Messages

CPF1618 E Job description &1 in library &2 damaged.
CPF24B4 E Severe error occurred while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C21 E Format name &1 not valid.
CPF3C24 E Length of receiver variable not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9804 E Object &2 in library &3 damaged.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.

Retrieve Job Information (QUSRJOBI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified job name	Input	Char(26)
5	Internal job identifier	Input	Char(16)

Optional Parameter:

6	Error code	I/O	Char(*)

The Retrieve Job Information (QUSRJOBI) API retrieves specific information about a job.

Authorities and Locks

Job Authority *JOBCTL, if the job for which information is retrieved has a different user profile from that of the job that calls the QUSRJOBI API.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. You can specify the size of this area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the job information to be returned. The format names supported are:

JOBI0100	Basic performance information
JOBI0150	Additional performance information
JOBI0200	WRKACTJOB information
JOBI0300	Job queue and output queue information
JOBI0400	Job attribute information
JOBI0500	Message logging information
JOBI0600	Active job information
JOBI0700	Library list information

Refer to "Selecting a Job Information Format" for details of each of the formats.

Qualified job name
INPUT; CHAR(26)
The name of the job for which information is to be returned. The qualified job name has three parts:

Job name CHAR(10). A specific job name or one of the following special values:
- * The job that this program is running in. The rest of the qualified job name parameter must be blank.
- ***INT** The internal job identifier locates the job. The user name and job number must be blank.

User name CHAR(10). A specific user profile name, or blanks when the job name is a special value or *INT.

Job number CHAR(6). A specific job number, or blanks when the job name specified is a special value or *INT.

If only the job name and no other qualifying values are given, the system searches all the jobs on the system for the job name. If the system finds duplicates of the specified job, it displays a list of messages containing the qualified job names of all duplicates.

Internal job identifier
INPUT; CHAR(16)
The internal identifier for the job. The List Job API, QUSLJOB, creates this identifier. If you do not specify *INT for the job name parameter, this parameter must contain blanks. With this parameter, the system can locate the job more quickly than with a job name.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Selecting a Job Information Format

The following section presents some of the performance characteristics of the different formats (primarily JOBI0100, JOBI0150, and JOBI0200). When formats return some of the same information, the performance effects are discussed. When a format contains information not available in other formats, performance is not discussed.

JOBI0100 This format returns basic performance information about a job. It is faster than the JOBI0150 format and the JOBI0200 format (which also contain performance information). The reason that this format is faster is that it does not touch as many objects, causing less paging when retrieving information about the job.

JOBI0150 This format returns additional performance information, and is slower than the JOBI0100 format. It is similar to the JOBI0200 format, but is faster than that format because there is less paging involved in retrieving the information.

JOBI0200 This format returns information equivalent to that found on the Work with Active Jobs (WRKACTJOB) command.

JOBI0300 This format returns job queue and output queue information for a job, as well as information about the submitter's job if the job is a submitted batch job.

JOBI0400 This format primarily returns job attribute types of information, but has other types of information as well.

JOBI0500 This format returns message logging information.

JOBI0600 This format returns information about active jobs only. It is intended to supplement the JOBI0400 format. It retrieves information from several additional objects associated with the job, and therefore, it causes additional paging.

JOBI0700 This format returns library list information for an active job.

Each format returns information that is only valid for the status of certain jobs. For example, the JOBI0200 format only returns information for active jobs. Because the job status can change between the time the list is generated and the time the Retrieve Job Information API is called, you must design your application to handle this.

When requesting information about a job that has an unknown or incorrect job status for the format requested, the API returns the current status of the job and sets the remainder of the fields for that format to zeros and blanks. When requesting information about a job that is not valid, the API returns the job's status as blanks and sets the remainder of the fields for that format to zeros and blanks. Therefore, you should check the returned status of the job before processing the data. Each format description specifies each status for which the API returns complete information.

JOBI0100 Format

The JOBI0100 format information is valid for active jobs and jobs on queues. For jobs on queues, this format returns zeros or blanks for the attributes. If the Change Job (CHGJOB) command was run against a job on a *JOBQ, the attributes returned are the attributes specified on the CHGJOB command. If the job status changes to *OUTQ, the status field returned is *OUTQ and the API returns no information other than the number of bytes returned, the number of bytes available, the qualified job name, and the internal job identifier.

The JOBI0100 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(2)	Reserved
64	40	BINARY(4)	Run priority
68	44	BINARY(4)	Time slice
72	48	BINARY(4)	Default wait
76	4C	CHAR(10)	Purge

For more details about the fields listed in the previous table, see "Field Descriptions" on page 37-20.

JOBI0150 Format

The JOBI0150 format is valid for active jobs only. If the job status changes to *OUTQ or *JOBQ, the status field is set appropriately, and no information other than the number of bytes returned, the number of bytes available, the qualified job name, and the internal job identifier are returned.

The JOBI0150 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format JOBI0100
86	56	CHAR(10)	Time slice end pool
96	60	BINARY(4)	Processing unit used
100	64	BINARY(4)	System pool identifier

For more details about the fields listed in the previous table, see "Field Descriptions" on page 37-20.

JOBI0200 Format

The JOBI0200 format is only valid for active jobs and is similar to the information supported by the Work with Active Jobs (WRKACTJOB) command. If the job status has changed to *OUTQ or *JOBQ, the status field is set appropriately, and no information other than the number of bytes returned, the number of bytes available, the qualified job name, and the internal job identifier is returned.

The JOBI0200 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(10)	Subsystem description name
72	48	BINARY(4)	Run priority
76	4C	BINARY(4)	System pool identifier
80	50	BINARY(4)	Processing unit used
84	54	BINARY(4)	Number of auxiliary I/O requests
88	58	BINARY(4)	Number of interactive transactions
92	5C	BINARY(4)	Response time total
96	60	CHAR(1)	Function type

Offset		Type	Field
Dec	Hex		
97	61	CHAR(10)	Function name
107	6B	CHAR(4)	Active job status

For more details about the fields listed in the previous table, see "Field Descriptions" on page 37-20.

JOBI0300 Format

This format returns job queue and output queue information for a job, as well as information about the submitter's job. This information is valid for any job status. The JOBI0300 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(10)	Job queue name
72	48	CHAR(10)	Job queue library name
82	52	CHAR(2)	Job queue priority
84	54	CHAR(10)	Output queue name
94	5E	CHAR(10)	Output queue library name
104	68	CHAR(2)	Output queue priority
106	6A	CHAR(10)	Printer device name
116	74	CHAR(10)	Submitter's job name
126	7E	CHAR(10)	Submitter's user name
136	88	CHAR(6)	Submitter's job number
142	8E	CHAR(10)	Submitter's message queue name
152	98	CHAR(10)	Submitter's message queue library name

For more details about the fields listed in the previous table, see "Field Descriptions" on page 37-20.

JOBI0400 Format

This format primarily returns job attribute types of information, but has other types of information as well. This format is valid for any job status. The JOBI0400 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(13)	Date and time job entered the system
75	4B	CHAR(13)	Date and time job became active
88	58	CHAR(15)	Job accounting code
103	67	CHAR(10)	Job description name
113	71	CHAR(10)	Job description library name
123	7B	CHAR(24)	Unit of work ID
147	93	CHAR(8)	Mode name
155	9B	CHAR(10)	Inquiry message reply
165	A5	CHAR(10)	Logging of CL programs
175	AF	CHAR(10)	Break message handling
185	B9	CHAR(10)	Status message handling
195	C3	CHAR(13)	Device recovery action
208	D0	CHAR(10)	DDM conversation handling
218	DA	CHAR(1)	Date separator
219	DB	CHAR(4)	Date format
223	DF	CHAR(30)	Print text
253	FD	CHAR(10)	Submitter's job name
263	107	CHAR(10)	Submitter's user name
273	111	CHAR(6)	Submitter's job number
279	117	CHAR(10)	Submitter's message queue name
289	121	CHAR(10)	Submitter's message queue library name
299	12B	CHAR(1)	Time separator
300	12C	BINARY(4)	Coded character set ID
304	130	CHAR(8)	Date and time job is scheduled to run
312	138	CHAR(10)	Print key format

For more details about the fields listed in the previous table, see "Field Descriptions" on page 37-20.

JOBI0500 Format

This format returns message logging information. This format is valid for any job status. The JOBI0500 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(2)	Reserved
64	40	BINARY(4)	End severity
68	44	BINARY(4)	Logging severity
72	48	CHAR(1)	Logging level
73	49	CHAR(10)	Logging text

For more details about the fields listed in the previous table, see "Field Descriptions" on page 37-20.

JOBI0600 Format

The JOBI0600 format returns information about active jobs. If the job status changes to *JOBQ or *OUTQ, the status field is set appropriately, and no information other than the number of bytes returned, the number of bytes available, the qualified job name, and the internal job identifier are returned.

The JOBI0600 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(8)	Job switches
70	46	CHAR(1)	End status
71	47	CHAR(10)	Subsystem description name
81	51	CHAR(10)	Subsystem description library name
91	58	CHAR(10)	Current user profile
101	65	CHAR(1)	DBCS capable

For more details about the fields listed in the previous table, see "Field Descriptions" on page 37-20.

JOBI0700 Format

The JOBI0700 format returns library list information for active jobs only. The format returns the actual length instead of the total length because all libraries may not exist. The JOBI0700 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(2)	Reserved
64	40	BINARY(4)	Number of libraries in SYSLIBL
68	44	BINARY(4)	Number of product libraries
72	48	BINARY(4)	Current library existence
76	4C	BINARY(4)	Number of libraries in USRLIBL
See note	See note	Array(*) of CHAR(11)	System library list (for each library in the list)
See note	See note	Array(*) of CHAR(11)	Product libraries if they exist
See note	See note	Array(*) of CHAR(11)	Current library if one exists
See note	See note	Array(*) of CHAR(11)	User library list (for each library in the list)

Note: The decimal and hexadecimal offsets depend on the number of libraries you have in the various parts of your library lists. The data is left-justified with a blank pad at the end. The array is sequential. It is an array or data structure. See the *CL Programmer's Guide* for the total number of libraries that can be returned to you.

For more details about the fields listed in the previous table, see "Field Descriptions" on page 37-20.

Field Descriptions

The following section describes the fields returned in further detail. The fields are listed in alphabetical order.

Active job status. The status of the job. Each job displays only one status. The possible values are:

no status A blank status field represents a job that is in transition or is not active.
BSCA Waiting in a pool activity level for the completion of an I/O operation to a binary synchronous device.
BSCW Waiting for the completion of an I/O operation to a binary synchronous device.
CMNA Waiting in a pool activity level for the completion of an I/O operation to a communications device.
CMNW Waiting for the completion of an I/O operation to a communications device.
CMTW Waiting for the completion of save-while-active checkpoint processing in another job.
CPCW Jobs waiting for the completion of a CPI Communications call.
DEQA Waiting in the pool activity level for completion of a dequeue operation.
DEQW Waiting for completion of a dequeue operation. For example, QSYSARB and subsystem monitors generally wait for work by waiting for a dequeue operation.
DKTA Waiting in a pool activity level for the completion of an I/O operation to a diskette unit.
DKTW Waiting for the completion of an I/O operation to a diskette unit.
DLYW The Delay Job (DLYJOB) command delays the job for a time interval to end, or for a specific delay end time. The function field shows either the number of seconds the job is to delay (999999), or the specific time when the job is to resume running.
DSC Disconnected from a work station display.
DSPA Waiting in a pool activity level for input from a work station display.
DSPW Waiting for input from a work station display.
END The job has been ended with the *IMMED option, or its delay time has ended with the *CNTRLD option.
EOFA Waiting in the activity level to try a read operation again on a database file after the end-of-file has been reached.
EOFW Waiting to try a read operation again on a database file after the end-of-file has been reached.
EOJ Ending for a reason other than running the End Job (ENDJOB) or End Subsystem (ENDSBS) command, such as SIGNOFF, End Group Job (ENDGRPJOB), or an exception that is not handled.
EVTW Waiting for an event. For example, QLUS and SCPF generally wait for work by waiting for an event.
GRP Suspended by a Transfer Group Job (TFRGRPJOB) command.
HLD Held.
ICFA Waiting in a pool activity level for the completion of an I/O operation to an intersystem communications function file.
ICFW Waiting for the completion of an I/O operation to an intersystem communications function file.
INEL Ineligible and not currently in the pool activity level.
LCKW Waiting for a lock.
MLTA Waiting in a pool activity level for the completion of an I/O operation to multiple files.
MLTW Waiting for the completion of an I/O operation to multiple files.
MSGW Waiting for a message from a message queue.
MXDW Waiting for the completion of an I/O operation to a mixed device file.
OSIW Jobs waiting for the completion of an OSI Communications Subsystem/400 OSLISN, OSRACS, OSRACA, OSRCV, or OSRCVA operation.
PRTA Waiting in a pool activity level for output to a printer to complete.
PRTW Waiting for output to a printer to be completed.
PSRW A prestart job waiting for a program start request.
RUN Currently running in the pool activity level.
SRQ The suspended half of a system request job pair.
SVFA Waiting in a pool activity level for completion of a save file operation.
SVFW Waiting for completion of a save file operation.
TAPA The job is waiting in a pool activity level for completion of an I/O operation to a tape device.
TAPW Waiting for completion of an I/O operation to a tape device.
TIMA Waiting in a pool activity level for a time interval to end.
TIMW Waiting for a time interval to end.

Break message handling. How this job handles break messages. The possible values are:

**NORMAL* The message queue status determines break message handling.
**HOLD* The message queue holds break messages until a user or program requests them. The work station user uses the Display Message (DSPMSG) command to display the messages; a program must issue a Receive Message (RCVMSG) command to receive a message and handle it.
**NOTIFY* The system notifies the job's message queue when a message arrives. For interactive jobs, the audible alarm sounds if there is one, and the message-waiting light comes on. For batch jobs, no notification occurs and the queue holds the message.

Coded character set ID. The coded character set identifier used for this job.

Current library if one exists. The name of the current library for the job. If no current library exists, the current

library existence field is zero and this field has no entry in the list.

Current library existence. The current library existence field:

0 No current library exists.
1 A current library exists.

Current user profile. The user profile that the job for which information is being retrieved is currently running under. This name may differ from the user portion of the job name.

Date and time job became active. When the job began to run on the system. This is blank if the job did not become active. It is in the format CYYMMDDHHMMSS, where:

C Century. 0 is the twentieth century, and 1 is the twenty-first century.
YY Year
MM Month
DD Day
HH Hour
MM Minute
SS Second

Date and time job entered system. When the job was placed on the system, in the CYYMMDDHHMMSS format described for the date and time job became active field.

Date and time job is scheduled to run. Date and time the job is scheduled to become active. This field is blank if the job is not a scheduled job. The format for this field is the system time-stamp format.

Date format. The format that the date is presented in. The following values are possible:

**YMD* Year, month, and day format
**MDY* Month, day, and year format
**DMY* Day, month, and year format
**JUL* Julian format (year and day)

Date separator. The value used to separate days, months, and years when presenting a date.

DBCS capable. Whether the job is DBCS capable.

0 The job is not DBCS capable.
1 The job is DBCS capable.

DDM conversation handling. Whether the distributed data management (DDM) conversations are *KEEP or *DROP when they are not used.

Default wait. The default maximum time (in seconds) that a job waits for a system instruction that performs a wait function, such as a LOCK machine interface (MI) instruction, to complete running.

Device recovery action. The action taken for interactive jobs when an I/O error occurs for the job's requesting program device. The possible values are:

**MSG* Signals the I/O error message to the application and lets the application program perform error recovery.
**DSCMSG* Disconnects the job when an I/O error occurs. When the job reconnects, the system sends an error message to the application program, indicating the job has reconnected and that the work station device has recovered.
**DSCENDRQS* Disconnects the job when an I/O error occurs. When the job reconnects, the system sends the End Request (ENDRQS) command to return control to the previous request level.
**ENDJOB* Ends the job when an I/O error occurs. A message is sent to the job's log and to the history log (QHST) indicating the job ended because of a device error.
**ENDJOBNOLIST* Ends the job when an I/O error occurs. There is no job log produced for the job. The system sends a message to the QHST log indicating the job ended because of a device error.

End severity. The message severity level of escape messages that can cause a batch job to end. The batch job ends when a request in the batch input stream sends an escape message, whose severity is equal to or greater than this value, to the request processing program.

End status. Whether or not the system issued a controlled cancelation. The possible values are:

1 The system, the subsystem in which the job is running, or the job itself is canceled.
0 The system, subsystem, or job is not canceled.
blank The job is not running.

Function name. Additional information (as described in the function type field) about the function the job is currently performing. This information is updated only when a command is processed.

Function type. Whether the job is performing a high-level function and what the function type is. The possible values are:

blank The system is not doing a logged function.
C A command is running interactively, or it is in a batch input stream, or it was requested from a system menu. Commands in CL programs or REXX procedures are not logged.
D The job is processing a Delay Job (DLYJOB) command. The function name contains the number of seconds the job is delayed (up to 999999 seconds), or the time when the job is to resume processing (HH:MM:SS), depending on how you specified the command.
G The Transfer Group Job (TFRGRPJOB) command suspended the job. The function name field contains the group job name for that job.
I The job is rebuilding an index (access path). The function name field contains the name of the logical file whose index is rebuilt.

L The system logs history information in a database file. The function name field contains the name of the log (QHST is the only log currently supported).

M The job is currently at a system menu. The function name field contains the name of the menu.

N The job is a multiple requester terminal (MRT) job if the job type is BATCH and the subtype is MRT, or it is an interactive job attached to an MRT job if the job type is interactive. See job type, subtype, or field for how to determine what type of job this is.

For MRT jobs, the function name field contains information in the following format:

- BINARY(4): The number of requesters currently attached to the MRT job.
- BINARY(4): The maximum number (MRTMAX) of requesters.
- CHAR(2): The never-ending program (NEP) indicator. If an MRT is also an NEP, the MRT stays active even if there are no requesters of the MRT. A value of NP indicates a never-ending program. A value of blanks indicates that it is not a never-ending program.

For interactive jobs attached to an MRT, the function name field contains the name of the MRT procedure:

O The job is a subsystem monitor that is performing input/output (I/O) operations to a work station. The function name field contains the name of the work station device to which the subsystem is performing an input/output operation.

P The job is running a program. The function name field contains the name of the program.

R The job is running a procedure. The function name field contains the name of the procedure.

* This does a special function. For this value, the function name field contains one of the following values:

- ADLACTJOB: Auxiliary storage is being allocated for the number of active jobs specified in the QADLACTJ system value. This may indicate that the system value for the initial number of active jobs is too low.
- ADLTOTJOB: Auxiliary storage is being allocated for the number of jobs specified in the QADLTOTJ system value.
- CMDENT: The Command Entry display is being used.
- DIRSHD: Directory shadowing.
- DLTSPLF: The system is deleting a spooled file.
- DUMP: A dump is in process.
- JOBLOG: The system is producing a job log.
- PASSTHRU: The job is a pass-through job.
- RCLSPLSTG: Empty spooled database members are being deleted.
- SPLCLNUP: Spool cleanup is in process.

Internal job identifier. A value input to other APIs to increase the speed of locating the job on the system. Only APIs described in this manual use this identifier. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Inquiry message reply. How the job answers inquiry messages:

**RQD* The job requires an answer for any inquiry messages that occur while this job is running.

**DFT* The system uses the default message reply to answer any inquiry messages issued while this job is running. The default reply is either defined in the message description or is the default system reply.

**SYSRPYL* The system reply list is checked to see if there is an entry for an inquiry message issued while this job is running. If a match occurs, the system uses the reply value for that entry. If no entry exists for that message, the system uses an inquiry message.

Job accounting code. An identifier assigned to the job by the system to collect resource use information for the job when job accounting is active.

Job description name. A set of job-related attributes used for one or more jobs on the system. These attributes determine how the job is run on the system. Multiple jobs can also use the same job description.

Job description library name. The library containing the job description.

Job name. The name of the job as identified to the system. For an interactive job, the system assigns the job the name of the work station where the job started; for a batch job, you specify the name in the command when you submit the job.

Job number. The system-generated job number.

Job queue name. The name of the job queue that the job is currently on, or that the job is on when it is currently active. This value is for jobs whose status is *JOBQ or *ACTIVE. For jobs with a status of *OUTQ, the value for this field is blank.

Job queue library name. The name of the library where the job queue is located.

Job queue priority. The scheduling priority of the job compared to other jobs on the same job queue. The highest priority is 0 and the lowest is 9. This value is for jobs

whose status is *JOBQ or *ACTIVE. For jobs with a status of *OUTQ, the value for this field is blank.

Job status. The status of the user jobs included in the list. The special values are:

*ACTIVE Active jobs. This includes group jobs, system request jobs, and disconnected jobs.
*ALL All jobs, regardless of status.
*JOBQ Jobs that are currently on job queues.
*OUTQ Jobs that have completed running but still have output on an output queue.

Job subtype. Additional information about the job type (if any exists). The possible values are:

blank The job has no special subtype or is an invalid job.
D The job is an immediate job.
E The job started with a procedure start request.
J The job is a prestart job.
P The job is a print driver job.
T The job is a System/36 multiple requester terminal (MRT) job.
U Alternate spool user.

Job switches. The current setting of the job switches used by this job. This value is for jobs whose status is *ACTIVE. For jobs with a status of *OUTQ or *JOBQ, the value is blank.

Job type. The type of job. The possible values for this field are:

blank The job is an invalid job.
A The job is an autostart job.
B The job is a batch job.
I The job is an interactive job.
M The job is a subsystem monitor job.
R The job is a spooled reader job.
S The job is a system job.
W The job is a spooled writer job.
X The job is the SCPF system job.

Refer to "Comparing Job Type and Subtype with the Work with Active Job Command" on page 37-25 for information about how the job type field and the job subtype field equate to the type field in the Work with Active Job (WKRACTJOB) command.

Logging of CL programs. Whether or not messages are logged for CL programs that are run. The possible values are *YES and *NO.

Logging level. What type of information is logged. The possible values are:

0 No messages are logged.
1 All messages sent to the job's external message queue with a severity greater than or equal to the message logging severity are logged.
2 The following information is logged:
- Level 1 information
- Requests or commands from CL programs for which the system issues messages with a severity code greater than or equal to the logging severity
- All messages associated with those requests or commands that have a severity code greater than or equal to the logging severity

3 The following information is logged:
- Level 1 information
- All requests or commands from CL programs
- All messages associated with those requests or commands that have a severity greater than or equal to the logging severity

4 The following information is logged:
- All requests or commands from CL programs
- All messages with a severity code greater than or equal to the logging severity

Logging severity. The minimum severity level that causes error messages to be logged in the job log.

Logging text. The level of message text that is written in the job log or displayed to the user when an error message is created according to the first two message logging values. The possible values are:

*MSG Only the message is written to the job log.
*SECLVL Both the message and the message help for the error message is written to the job log.
*NOLIST If the job ends normally, there is no job log. If the job ends abnormally (if the job end code is 20 or higher), there is a job log. The messages appearing in the job's log contain both the message and the message help.

Mode name. The mode name of the advanced program-to-program communications device that started the job.

Number of auxiliary I/O requests. The number of auxiliary I/O requests for the job. This includes both database and non-database paging.

Number of bytes available. All of the available bytes for use in your application.

Note: When you request format JOBI0700, the actual length depends on how many libraries are in the library list.

Number of bytes returned. The number of bytes returned to the user. This may be some but not all of the bytes available.

Number of interactive transactions. The count of operator interactions, such as pressing the Enter key or a function key. This field is zero for jobs that have no interactions.

Number of libraries in SYSLIBL. The number of libraries in the system part of the library list.

Number of libraries in USRLIBL. The number of libraries in the user library list.

Number of product libraries. The number of product libraries found in the library list.

Output queue name. The name of the default output queue that is used for spooled output produced by this job. The default output queue is only for spooled printer files that specify *JOB for the output queue.

Output queue library name. The name of the library containing the output queue.

Output queue priority. The output priority for spooled output files that this job produces. The highest priority is 0, and the lowest is 9.

Print key format. Whether border and header information is provided when the Print key is pressed.

**NONE* The border and header information is not included with output from the Print key.
**PRTBDR* The border information is included with output from the Print key.
**PRTHDR* The header information is included with output from the Print key.
**PRTALL* The border and header information is included with output from the Print key.

Printer device name. The printer device used for printing output from this job.

Print text. The line of text (if any) that is printed at the bottom of each page of printed output for the job.

Processing unit used. The amount of processing unit time (in milliseconds) that the job used. A value of -1 is returned if the field is not large enough to hold the actual result. This has the same meaning as the + + + signs on the Work with Active Jobs display.

Product libraries if they exist. The library that contains product information. A blank is in the last position of the name.

Purge. Whether or not this job is eligible to be moved out of main storage and put into auxiliary storage at the end of a time slice or when it is beginning a long wait (such as waiting for a work station user's response). The possible values are:

**YES* The job is eligible to be moved out of main storage and put into auxiliary storage.
**NO* The job is not moved out of main storage. However, when some of main storage is needed to run other jobs in the same storage pool, pages belonging to this job may be moved (eight at a time) to auxiliary storage to accommodate pages needed by other jobs. When this job runs again, its pages are returned to main storage as they are needed.
blank Not used for job types *JOBQ or *OUTQ, or for invalid jobs.

Reserved. An ignored field.

Response time total. The total amount of response time for the job, in milliseconds. This value does not include the time used by the machine, attached input/output (I/O) hardware, and transmission lines for sending and receiving data. This field is zero for jobs that have no interactions. A value of -1 is returned if the field is not large enough to hold the actual result.

Run priority. The priority at which the job is currently running, relative to other jobs on the system. The run priority ranges from 1 (highest) to 99 (lowest).

Status message handling. Whether you want status messages displayed for this job. The possible values are:

**NONE* This job does not display status messages.
**NORMAL* This job displays status messages.

Submitter's job name. The job name of the submitter's job. If the job has no submitter, this field is blank.

Submitter's job number. The job number of the submitter's job. If the job has no submitter, this field is blank.

Submitter's message queue library name. The name of the library that contains the message queue. If the job has no submitter, this field is blank.

Submitter's message queue name. The name of the message queue where the system sends a completion message when a batch job ends. If the job has no submitter, this field is blank.

Submitter's user name. The user name of the submitter. If the job has no submitter, this field is blank.

Subsystem description library name. The library that contains the subsystem description. This value is only for jobs whose status is *ACTIVE. For jobs with a status of *OUTQ or *JOBQ, the value for this field is blank.

Subsystem description name. The name of the subsystem in which an active job is running. This value is only for jobs whose status is *ACTIVE. For jobs with status of *OUTQ or *JOBQ, the value for this field is blank.

System library list (for each library in the list). The system portion of the job's library list. A blank is in the last position of the name.

System pool identifier. The identifier of the system-related pool from which the job's main storage is allocated. These identifiers are not the same as those specified in the subsystem description, but are the same as the system pool identifiers shown on the system status display.

Time separator. The value used to separate hours, minutes, and seconds when presenting a time.

Time slice. The maximum amount of processor time (in milliseconds) given to this job before other jobs are given the opportunity to run. The time slice establishes the amount of time needed by the job to accomplish a meaningful amount of processing. At the end of the time slice, the job might be put in an inactive state so that other jobs can become active in the storage pool.

Time slice end pool. Whether you want interactive jobs moved to another main storage pool at the end of the time slice. The possible values are:

**NONE* The job does not move to another main storage pool when it reaches the end of the time slice.

**BASE* The job moves to the base pool when it reaches the end of the time slice.

Unit of work ID. The unit of work ID is used to track jobs across multiple systems. If a job is not associated with a source or target system using advanced program-to-program communications (APPC), this information is not used. Every job on the system is assigned a unit of work ID. The unit-of-work identifier is made up of:

Location name
CHAR(8). The name of the source system that originated the APPC job.

Network ID
CHAR(8). The network name associated with the unit of work.

Instance
CHAR(6). The value that further identifies the source of the job. This is shown as hexadecimal data.

Sequence number
CHAR(2). A value that identifies a checkpoint within the application program.

User library list (for each library in the list). The user portion of the job's library list. A blank is in the last position of the name.

User name. The user profile under which the job runs. The user name is the same as the user profile name and can come from several different sources depending on the type of job.

Comparing Job Type and Subtype with the Work with Active Job Command

The following table compares the job type and job subtype fields returned by the QUSRJOBI API to the type field on the Work with Active Job (WRKACTJOB) command.

Figure 37-1. WRKACTJOB and QUSRJOBI API Comparison

Job Type Field	Job Type	Job Subtype
ASJ (Autostart)	A	blank
BCH (Batch)	B	blank
BCI (Batch immediate)	B	D
EVK (Started by a program start request)	B	E
INT (Interactive)	I	blank
MRT (Multiple requestor terminal)	B	T
PJ (Prestart job)	B	J
PDJ (Print driver job)	W	P
RDR (Reader)	R	blank
SYS (System)	S or X	blank
SBS (Subsystem monitor)	M	blank
WTR (Writer)	W	blank
blank (Alternative user subtype—not an active job)	blank	U

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C20 E Error found by program &1.
CPD3C21 D Format name &1 is not valid.
CPD3C24 D Length of the receiver variable is not valid.
CPD3C51 D Internal identifier is not blanks and job name is not *INT.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF3C51 E Internal job identifier not valid.
CPF3C52 E Internal job identifier no longer valid.
CPF3C53 E Job &3/&2/&1 not found.
CPF3C54 E Job &3/&2/&1 currently not available.
CPF3C55 E Job &3/&2/&1 does not exist.
CPF3C57 E Not authorized to retrieve job information.
CPF3C58 E Job name specified is not valid
CPF3C59 E Internal identifier is not blanks and job name is not *INT.
CPF9820 E Not authorized to use library &1.

Retrieve Job Queue Information (QSPRJOBQ) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Job queue name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Job Queue Information (QSPRJOBQ) API retrieves information associated with a specified job queue.

Authorities and Locks

Job Queue Library Authority
The caller needs *READ authority to the job queue library.

Job Queue Authority

The caller needs one of the following:

- *READ authority to the job queue.
- Job control special authority (*JOBCTL) if the job queue is operator controlled (OPRCTL(*YES)).
- Spool control special authority (*SPLCTL).

Job Queue Lock

This API gets an *EXCLRD lock on the job queue.

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The variable that is to receive the information requested.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable provided by the receiver variable parameter. The amount of data returned is truncated if it is too small. A length of less than 8 is not valid.

Format name

INPUT; CHAR(8)

The content and format of the queue information being returned. The JOBQ0100 format must be used for the job queue information. See "JOBQ0100 Format" to view the information returned for this format.

Job queue name

INPUT; CHAR(20)

The name of the job queue for which information is returned. The first 10 characters contain the queue name, and the second 10 characters contain the name of the library in which the queue resides.

The following special values are supported for the library name:

***LIBL** The library list used to locate the job queue.

***CURLIB** The current library for the job is used to locate the job queue.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

JOBQ0100 Format

The following table shows the information returned for the JOBQ0100 format. For more details about the fields in the following table see, "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Job queue name
18	12	CHAR(10)	Job queue library name
28	1C	CHAR(10)	Operator controlled
38	26	CHAR(10)	Authority to check
48	30	BINARY(4)	Number of jobs
52	34	CHAR(10)	Job queue status
62	3E	CHAR(10)	Subsystem name
72	48	CHAR(50)	Text description

Field Descriptions

Authority to check. Whether the user must be the owner of the queue in order to control the queue by holding or releasing the queue. The possible values are:

**OWNER* Only the owner of the job queue can control the queue.

**DTAAUT* Any user with *READ, *ADD, or *DELETE authority to the job queue can control the queue.

Bytes available. Total format data length.

Bytes returned. Length of the data returned.

Job queue library name. The name of the library that contains the job queue.

Job queue name. The name of the job queue.

Job queue status. The status of the job queue. The status may be one of the following values:

RELEASED The queue is released.

HELD The queue is held.

Number of jobs. The number of jobs in the queue.

Operator controlled. Whether a user who has job control authority is allowed to control this job queue and manage the jobs on the queue.

**YES* Users with job control authority can control the queue and manage the jobs on the queue.

**NO* This queue and its jobs cannot be controlled by users with job control authority unless they also have other special authority.

Subsystem name. The name of the subsystem that can receive jobs from this job queue. If there is no name, then this queue is not associated with an active subsystem, and no job can be processed.

Text description. Text that briefly describes the job queue.

**BLANK* There is no text description of the job queue.

Error Messages

CPF2207 E Not authorized to use object &1 in library &3 type &2.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF3307 E Job queue &1 in &2 not found.
CPF3330 E Necessary resource not available.
CPF8121 E &8 damage on job queue &4 in &9.
CPF8122 E &8 damage on library &4.

Retrieve Subsystem Information (QWDRSBSD) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Subsystem and library name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Subsystem Information (QWDRSBSD) API retrieves information about a specific subsystem.

Authorities and Locks

Subsystem Description Authority *USE
Library Authority *READ

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable to receive the subsystem information. This area must be large enough to accommodate the information specified.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. The length must be at least 8 bytes. If the variable is not long enough to hold the subsystem information, the data is truncated.

Format name
INPUT; CHAR(8)
The format of the subsystem information. You can use this format:

SBSI0100 Basic subsystem information. For details, see "SBSI0100 Format."

Subsystem and library name
INPUT; CHAR(20)
The subsystem about which to retrieve information, and the library in which the subsystem description is located. The first 10 characters contain the subsystem name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

SBSI0100 Format

The following table shows the information returned in the receiving variable for the SBSI0100 format. For a detailed description of each field, see "Field Descriptions" on page 37-28.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Subsystem description name
18	12	CHAR(10)	Subsystem description library name
28	1C	CHAR(10)	Subsystem status
38	26	CHAR(10)	Sign-on device file name
48	30	CHAR(10)	Sign-on device file library
58	3A	CHAR(10)	Secondary language library
68	44	BINARY(4)	Maximum active jobs
72	48	BINARY(4)	Currently active jobs
76	4C	BINARY(4)	Number of storage pools defined
Offsets vary. These five fields repeat, in the order listed, for each pool defined for the subsystem.		BINARY(4)	Pool ID
		CHAR(10)	Pool name
		CHAR(6)	Reserved
		BINARY(4)	Pool size
		BINARY(4)	Pool activity level

Field Descriptions

Bytes available. The total length of all data available.

Bytes returned. The length of the data actually returned. The number of bytes returned is always less than or equal to both the number of bytes available and the receiving variable length.

Currently active jobs. The number of jobs currently active in the subsystem. This number *includes* held jobs but *excludes* jobs that are disconnected or suspended because of a transfer secondary job or a transfer group job. If the subsystem status is *INACTIVE, this number is 0.

Maximum active jobs. The maximum number of jobs that can run or use resources in the subsystem at one time. If the subsystem description specifies *NOMAX, indicating that there is no maximum, this number is -1.

Number of storage pools defined. The number of storage pools defined for the subsystem. The maximum number of storage pools for a subsystem is currently 10. This number determines how many times the pool ID, pool name, pool size, pool activity level, and reserved fields are repeated. Those five fields are repeated as a group for each pool defined for the subsystem.

Pool activity level. The maximum number of jobs that can be active in the pool at one time. If the pool name indicates a system-defined pool, the number returned is 0.

Pool ID. The pool ID for the subsystem pool.

Pool name. The name of the subsystem pool. If the pool is user-defined, the value of this field is *USERPOOL. If the pool is system-defined, the value is one of these names:

**BASE*
: The system base pool, which can be shared with other subsystems. The QBASPOOL system value defines the base pool's size. The QBASACTLVL system value defines its activity level.

**INTERACT*
: The shared pool used for interactive work.

**NOSTG*
: No storage size or activity level is assigned to this storage pool.

**SHRPOOL1 – *SHRPOOL10*
: Shared pools.

**SPOOL*
: The shared pool for spooling writers.

The Change Shared Storage Pool (CHGSHRPOOL) command specifies the size and activity level of shared pools.

Pool size. If the pool name is *USERPOOL, the amount of storage, in kilobytes, that the pool attempts to allocate. If the pool has any other name, the value of this field is 0.

Reserved. An ignored field.

Secondary language library. The name of the subsystem's secondary language library.

Sign-on device file library. The name of the library in which the sign-on device file resides.

Sign-on device file name. The name of the sign-on file used by the subsystem.

Subsystem description library name. The name of the library in which the subsystem description resides.

Subsystem description name. The name of the subsystem about which information is being returned.

Subsystem status. The activity level of the subsystem. Valid values are *ACTIVE, indicating that the subsystem is running, and *INACTIVE, indicating that the subsystem is not running.

Error Messages

CPF1605	E	Cannot allocate subsystem description &1.
CPF1606	E	Error during allocation of subsystem &1.
CPF1607	E	Previous request pending for subsystem &1.
CPF1608	E	Subsystem description &1 not found.
CPF1619	E	Subsystem description &1 in library &2 damaged.
CPF1835	E	Not authorized to subsystem description.
CPF3CF1	E	Error code parameter not valid.
CPF3C21	E	Format name &1 is not valid.
CPF3C24	E	Length of the receiver variable is not valid.
CPF8122	E	&8 damage on library &4.
CPF9807	E	One or more libraries in library list deleted.
CPF9810	E	Library &1 not found.
CPF9820	E	Not authorized to use library &1.
CPF9830	E	Cannot assign library &1.

Set Lock Flight Recorder (QWTSETLF) API

Parameters

Required Parameter:

1	Set value	Input	Char(4)

Optional Parameter:

2	Error code	I/O	Char(*)

The Set Lock Flight Recorder (QWTSETLF) API turns the lock flight recorder on and off. The value of *ON is passed to the program to turn the lock flight recorder on, and *OFF is passed to turn the lock flight recorder off.

When the lock flight recorder is turned on, the system will begin logging successful lock operations on devices in the lock flight recorder for the device being locked.

This API should be used only when recommended by your IBM service representative.

Required Parameter

Set value

INPUT; CHAR(4)
The value passed to turn the lock flight recorder on or off. The valid values are:

***ON** Turn flight recorder on.
***OFF** Turn flight recorder off.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF119B E Value &1 specified for parameter is not valid.
CPF3CF1 E Error code parameter not valid.
CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.

Part 18. Work Station Support APIs

Chapter 38. Work Station Support APIs

The work station support APIs let you use the type-ahead and attention key buffering functions in your applications. In the past, type-ahead and attention key buffering were active by default for all interactive jobs on the system.

Type-ahead, also called **keyboard buffering**, lets the user type data faster than it can be sent to the system. **Attention key buffering** determines how to process the action of pressing an Attention key. If attention key buffering is on, the Attention key is treated as any other key. If attention key buffering is not on, pressing the Attention key results in sending the information to the system even when other work station input is inhibited.

The work station support APIs and their functions are:

Query Keyboard Buffering (QWSQRYWS) determines the current type-ahead and attention key buffering settings.

Set Keyboard Buffering (QWSSETWS) controls the use of the type-ahead and attention key buffering functions.

You can enter the parameters for the QWSQRYWS and QWSSETWS APIs in mixed case. The APIs convert them to uppercase. For all other APIs, you must enter all parameters in uppercase.

The keyboard buffering data stream is supported by the following controllers:

ASCII Work Station Input/Output Processor
Twinaxial Work Station Input/Output Processor
5394 Remote Control Unit

IBM Personal Computer Systems attached via PC Support/400 or work station emulation (WSE) do not support the type-ahead data stream. Keyboard buffering for these devices is controlled through the emulation programs.

When a callable API parameter is not valid or the controller does not support type-ahead, the exception is signaled to the caller from the work station. All other exceptions, including device allocation and authority errors, are returned to the caller.

Query Keyboard Buffering (QWSQRYWS) API

Parameters

Required Parameter Group:

1	Keyboard buffering	Output	Char(1)

Optional Parameter:

2	Device name	Input	Char(10)

The Query Keyboard Buffering (QWSQRYWS) API sends the data stream command to query the current value for keyboard buffering for a specified display.

Required Parameter Group

Keyboard buffering
OUTPUT; CHAR(1)
The current setting for keyboard buffering for the device. Valid values are:

- **0** The type-ahead and attention key buffering functions are off.
- **1** The type-ahead and attention key buffering functions are on.
- **2** The type-ahead function is on, and the attention key buffering function is off.

Optional Parameter

Device name
INPUT; CHAR(10)
The display to query for the keyboard buffering value. You can specify a particular device or use this special value:

***REQUESTER** The job's requesting program device is queried for the keyboard buffering value. This is the default.

Error Messages

CPF94FC E Type-ahead data stream not supported by controller.
CPF94FD E Type-ahead option parameter value not valid.

Set Keyboard Buffering (QWSSETWS) API

Parameters

Required Parameter Group:

1	Keyboard buffering	Input	Char(1)

Optional Parameter:

2	Device name	Input	Char(10)

The Set Keyboard Buffering (QWSSETWS) API controls the type-ahead and attention key buffering functions for a display. With the QWSSETWS API, you can:

- Turn both functions off
- Turn both functions on
- Turn on the type-ahead function without buffering the Attention key
- Send the data stream to a specific device

Any changes to the keyboard buffering value made through this program take effect immediately.

The keyboard buffering data stream is supported by the ASCII Work Station Input/Output Processor, Twinaxial Work Station Input/Output Processor, and 5394 Remote Control

Unit. Keyboard buffering for personal computer systems attached via PC Support/400 or work station emulation (WSE) is controlled through the emulation programs; these devices do not support the type-ahead data stream.

Required Parameter Group

Keyboard buffering
INPUT; CHAR(1)
The setting for keyboard buffering for a display. Valid values are:

0 The type-ahead and attention key buffering functions are off.
1 The type-ahead and attention key buffering functions are on.
2 The type-ahead function is on, and the attention key buffering function is off.

Optional Parameter

Device name
INPUT; CHAR(10)
The device to set the keyboard buffering value on. You can specify the name of a particular device or use this special value:

***REQUESTER** The keyboard buffering value is set on the job's requesting program device. This is the default.

Error Messages

CPF94FC E Type-ahead data stream not supported by controller.
CPF94FD E Type-ahead option parameter value not valid.

Part 19. Miscellaneous APIs

Chapter 39. Miscellaneous APIs

This chapter contains information about the Character Data Representation Architecture APIs. The APIs are:

Convert Graphic Character Strings (CDRCVRT)
Get CCSID for Normalization (CDRGCCN)
Get Encoding Scheme, Character Set, and Code Page Elements (CDRGESP)
Get Related Default CCSID (CDRGRDC)
Get Short Form (CCSID) from Specified ES, (CP,CS) (CDRSCSP)

These APIs are located in the QSYS2 library. More information on CCSIDs can be found in the *Character Data Representation Architecture (CDRA) Level 1 Reference* manual, SC09-1390.

This chapter also contains information about the following APIs:

Convert Date and Time Format (QWCCVTDT) allows you to convert date and time formats from one format to another format.
Remove All Bookmarks from a Course (QEARMVBM) allows you to remove the bookmarks from a Tutorial System Support course.
Retrieve Main Storage (QVTRMSTG) allows you to get at main storage and retrieve the same type of information that you receive from the Start System Service Tools (STRSST) command.

The APIs in this chapter are presented in alphabetical order.

Convert Date and Time Format (QWCCVTDT) API

Parameters

Required Parameter Group:

1	Input format	Input	Char(10)
2	Input variable	Input	Char(*)
3	Output format	Input	Char(10)
4	Output variable	Output	Char(*)
5	Error code	I/O	Char(*)

The Convert Date and Time Format (QWCCVTDT) API converts date and time values from one format to another format. The QWCCVTDT API lets you:

- Convert a time-stamp (*DTS, for system time-stamp) value to character format
- Convert a character date-time value to time-stamp format
- Convert a date from one character format to another
- Retrieve the current machine clock time and return it in the format you specify

Required Parameter Group

Input format
INPUT; CHAR(10)
The format of the data you give QWCCVTDT to convert. Valid values are:

***CURRENT** Current machine clock time.

***DTS** System time-stamp. *DTS date values range only from August 23, 1928, 12:03:06.315 to May 10, 2071, 11:56:53.684. Converting a date outside this range to *DTS format results in a date within this range. When you convert a character date-time value to *DTS and back to character format, there is a rounding error of plus or minus 2 milliseconds.

***JOB** The format given in the DATFMT job attribute.

***SYSVAL** The format given in the QDATFMT system value.

***YMD** YYMMDD (year, month, day) format.

***MDY** MMDDYY (month, day, year) format.

***DMY** DDMMYY (day, month, year) format.

***JUL** Julian format (YYDDD (year, day of year)).

You can convert any format except *CURRENT to the same format without receiving an error. When you convert a format to the same format, the input variable is copied into the output variable without validation.

When you convert one character date format (that is, anything other than *CURRENT, the current machine-clock time, or *DTS, the system time-stamp) to another character date format, the date information is validated and converted. However, the time and milliseconds portions of the input variable are copied into the output variable without validation.

Input variable
INPUT; CHAR(*)
The data to be converted. If the input format is *CURRENT, this parameter is ignored. If the input format is *DTS, the first 8 characters of the input variable are used. If the input format is one of the character formats (that is, anything other than *CURRENT or *DTS), the first 16 characters of the input variable are used; for details, see the following section, "Character Date and Time Value Structure."

Output format
INPUT; CHAR(10)
The format to convert the data to. Valid values are:

***DTS** System time-stamp
***JOB** The format given in the DATFMT job attribute
***SYSVAL** The format given in the QDATFMT system value
***YMD** YYMMDD format

***MDY** MMDDYY format
***DMY** DDMMYY format
***JUL** Julian format (YYDDD)

Output variable
OUTPUT; CHAR(*)
The converted data. If the output format is *DTS, the first 8 characters of this parameter are used. If the output format is one of the character formats (that is, anything other than *DTS), the first 16 characters of the output variable are used; for details, see "Character Date and Time Value Structure."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Character Date and Time Value Structure

This table shows the structure used for the input and output variables:

Offset	Description
0	The century digit, which must be one of these: **0** 20th century **1** 21st century
1–6	The date, left-justified. This value cannot be all blanks or all zeros. Left-justify Julian dates, using blanks to fill the space.
7–12	The time, in HHMMSS (hours, minutes, seconds) format.
13–15	The milliseconds. This value cannot be blanks.

Error Messages

CPF1060 E Date not valid.
CPF1061 E Time not valid.
CPF1848 E Century digit &1 not valid.
CPF1849 E Milliseconds value &1 not valid.
CPF1850 E Format &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.

Convert Graphic Character Strings (CDRCVRT) API

Parameters

Required Parameter Group:

1	CCSID1	Input	Binary(31)
2	ST1	Input	Binary(31)
3	S1	Input	Binary(31)
4	L1	Input	Binary(31)
5	CCSID2	Input	Binary(31)
6	ST2	Input	Binary(31)
7	GCCASN	Input	Binary(31)
8	L2	Input	Binary(31)
9	S2	Output	Binary(31)
10	L3	Output	Binary(31)
11	L4	Output	Binary(31)
12	FB	Output	Binary(31)

The Convert Graphic Character Strings (CDRCVRT) API is used by any program that needs an application programming interface to convert a graphic character data string of one type and one CCSID[1] to another string type and CCSID. The function assumes that the entire string to be converted is known and is passed to the function. This API will accept a *to* string type and CCSID along with a *from* string type and CCSID. The API then returns the converted graphic character data to the invoker.

The API makes the following checks on the input variables:

- The CCSID1 and CCSID2 values are verified. If either has an invalid value or out of range value, a feedback code will be returned to indicate an error.
- ST1, ST2, L1, L2, and GCCASN are verified for valid values. If an error condition is found, a feedback code will be returned to indicate an error.

On a successful return, this API returns the converted string as output and a feedback code of 0000,0000.

Required Parameter Group

CCSID1
INPUT; BINARY(31)
Variable containing the CCSID value for the input graphic character data string to be converted. The field type is 32-bit two's complement binary (a positive number from 1 to 65,534).

ST1
INPUT; BINARY(31)
Variable containing the string type of the input data string represented in the *from* CCSID. The field type is 32-bit two's complement binary (a positive number from 0 to 255). 0 and 1 are the only values currently used. All others are reserved.

0 Graphic character string, semantically defined by CCSID1
1 Graphic character string, semantically defined by

1 Coded character set identifier is a 16-bit number that identifies a specific set of encoding scheme identifier, character set identifier(s), code page identifier(s), and additional coding-related required information, that uniquely identifies the coded graphic character representation used.

CCSID1 and null terminated.[2]

S1 INPUT; BINARY(31)
The starting address of the area in the invoker's address space that contains the graphic character data to be converted.

L1 INPUT; BINARY(31)
Variable containing the length (in bytes) of the input string of character data contained in the area of the invoker's address space, starting at S1. The field type is 32-bit two's complement binary (positive number from 1 to 32,767).

If ST1 is 1, and the null-termination character is not encountered within L1 bytes, an error condition is indicated. The function then performs as if ST1 = 0.

CCSID2
INPUT; BINARY(31)
Variable containing the CCSID value for the converted graphic character data string. The field type is 32-bit two's complement binary (a positive number from 1 to 65,534).

ST2
INPUT; BINARY(31)

Variable containing the string type of the output data string represented in the *to* CCSID. The field type is 32-bit two's complement binary (a positive number from 0 to 255). 0, 1, and 2 are the only values currently used. All others are reserved.

- **0** Graphic character string, semantically defined by CCSID2
- **1** Graphic character string, semantically defined by CCSID2 and null terminated.
- **2** Graphic character string, semantically defined by CCSID2 and space padded.

GCCASN
INPUT; BINARY(31)

Variable containing a number that identifies which conversion alternative is selected to convert graphic character data from (CCSID1,ST1) to (CCSID2, ST2). Field type is 32-bit two's complement binary (a positive number from 0 to 255). 0 and 1 are the only values currently used. All others are reserved.

- **0** Indicates that the installation default conversion alternative is to be selected
- **1** Used to select IBM default mapping

Note: Currently, 0 and 1 both select the IBM default mapping.

L2 INPUT; BINARY(31)
Variable containing the length (in number of bytes) of the allocated area, starting at S2, to contain the converted graphic character data. The field type is 32-bit two's complement binary (a positive number from 1 to 32,767).

S2 OUTPUT; BINARY(31)
The starting address of the area in the invoker's address space that contains the graphic character data that has been converted. The area's length is given in L2.

If there is insufficient space in S2, a truncated converted string is returned in S2 and a feedback code is returned.

L3 OUTPUT; BINARY(31)

Variable containing the length (in number of bytes) of the converted string returned in S2. The field type is 32-bit two's complement binary (a positive number from 1 to 32,767).

L3 will always be ≤ L2. It includes any termination characters that are required by ST2.

- **ST2 = 1** Includes the number of bytes for null-terminator
- **ST2 = 2** Includes the number of bytes for space-padding

L4 OUTPUT; BINARY(31)

Variable containing a byte number in the input when the conversion process encounters an error condition. It points to the byte being processed when the error is detected. The field type is 32-bit two's complement binary (a positive number from 1 to 32,767).

When a conversion is error-free, a 0 is returned by this variable.

Note: The current implementation of this API does not support this function. This value will always be returned as 0.

FB OUTPUT; BINARY(31)
A feedback array containing the processing status (and any associated reason) returned by this function. an array of three 32-bit two's complement binary values (12 bytes or 96 bits). The status code is a positive number in the first 16 bits and the reason code is a positive number in the second 16 bits.

Feedback Codes

The feedback codes returned in hexadecimal by this function are:

0000 0000	Successful
0001 0001	No entry matching CCSID1/ST1 to CCSID2/ST2 using the alternative value in GCCASN
0002 0001	CCSID1 is invalid value of 0
0002 0002	CCSID2 is invalid value of 0
0003 0001	CCSID1 is invalid value of 65535

[2] A variable-length graphic character string, which is terminated by a character, whose code point has a binary value of zero. The number of bits in the code point used to represent the terminating character (the null terminator) is the smallest number of bits allowed for code points in the encoding scheme used.

0003 0002	CCSID2 is invalid value of 65535
0004 0001	L2 length for area S2 too small for output data
0005 0001	A DBCS graphic CCSID1 was specified, but a single byte code point was encountered
0005 0004	ES of CCSID1 is X'1301' and a malformed string was found
0005 0005	ST1 = 1, but no null-termination char found in input
0005 0006	ST2 = 1, but output string has more than one null-terminator characters
0005 0008	ST2 = 2 and ES is DBCS graphic, but an odd number of bytes are left in S2. Last byte of S2 is unpredictable. The conversion was done.
0005 0009	ST2 = 2 and ES is DBCS graphic, but an odd number of bytes are left in S2. Last byte of S2 is unpredictable. The conversion was done.
0007 0001	GCCST resource is invalid structure
0008 0001	CCSID1 value outside permitted range
0008 0002	CCSID2 value outside permitted range
0008 0003	ST1 outside permitted range - 0 is assumed
0008 0004	ST2 outside permitted range - 0 is assumed
0008 0005	L1 value outside permitted range (restricted to a maximum of 32,767)
0008 0006	L2 value outside permitted range (restricted to a maximum of 32,767)
0008 0007	GCCASN value is outside the permitted range - uses 0
0100 0002	One or more input characters were replaced with another character in CCSID2.

The following feedback codes are reserved:

0001 0002
0005 0002
0005 0003
0005 0007
0006 0001
0006 0002
0006 0003
0006 0004
0006 0005
0006 0007
0006 0008
0007 0002
0007 0003
0100 0001

Get CCSID for Normalization (CDRGCCN) API

Parameters

Required Parameter Group:

1	CCSID1	Input	Binary(31)
2	CCSID2	Input	Binary(31)
3	CCSIDn	Output	Binary(31)
4	HINTV	Output	Binary(31)
5	FB	Output	Array(3)

The Get CCSID for Normalization (CDRGCCN) API is used by any program that needs an application programming interface to determine the CCSID necessary for normalization.[3]

The API will make the following checks on the input variables:

- The CCSID1 and CCSID2 values are verified. If either has an invalid value or out of range value, a feedback code will be returned to indicate an error.

On a successful return, this API returns the third CCSID as output.

Required Parameter Group

CCSID1
INPUT; BINARY(31)
Variable containing the first CCSID value passed by the invoker. Field type is 32-bit two's complement binary (a positive number from 1 to 65,534).

CCSID2
INPUT; BINARY(31)
Variable containing the second CCSID value passed by the invoker. Field type is 32-bit two's complement binary (a positive number from 1 to 65,534).

CCSIDn
OUTPUT; BINARY(31)
Variable containing the returned CCSID value for normalization. Field type is 32-bit two's complement binary (a positive number from 1 to 65,534).

HINTV
OUTPUT; BINARY(31)
Variable that is a 32-bit two's complement binary number. Possible values are: 0, 1, 2, and 3. All other values are reserved. The function returns, in this variable, a number that conveys information to assist the invoking function in its subsequent processing.

- **0** No hints
- **1** CCSID1 and CCSID2 have the same CP.
- **2** CCSIDN has the same CP as CCSID2. Convert only CCSID1 to CCSIDN
- **3** CCSIDN has the same CP as CCSID1. Convert only CCSID2 to CCSIDN

[3] Normalization is necessary when certain operations such as concatenation or comparison are performed on graphic character strings. The two strings are both in the same CCSID, or they are normalized first to a single CCSID before the operation is done. The CDRGCCN API assists in determining the CCSID for normalization when given two CCSIDs. The returned CCSID may equal one or both the input CCSIDs.

FB OUTPUT; ARRAY(3)
A feedback array containing the processing status (and any associated reason) returned by this function. Field type is an array of three 32-bit two's complement binary values (12 bytes or 96 bits). The status code is a positive number in the first 16 bits and the reason code is a positive number in the second 16 bits.

Feedback Codes

The feedback codes returned, in hexadecimal, by this function are:

0000 0000	Successful
0001 0001	No entry for CCSID1/CCSID2 pair - (translation not defined)
0002 0001	CCSID1 has an invalid value of 0
0002 0002	CCSID2 has an invalid value of 0
0003 0001	CCSID1 has an invalid value of 65535
0003 0002	CCSID2 has an invalid value of 65535
0007 0001	NSCT repository is invalid in structure
0008 0000	CCSID1 value is outside permitted range
0008 0001	CCSID2 value is outside permitted range

These feedback codes are reserved:

0006 0001
0006 0002

Get Encoding Scheme (CDRGESP) API

Required Parameter Group:

Required Parameter Group:

1	CCSID1	Input	Binary(31)
2	N1	Input	Binary(31)
3	N2	I/O	Binary(31)
4	ES	Output	Binary(31)
5	CSCPL	Output	Array(N1)
6	FB	Output	Array(3)

The Get Encoding Scheme (CDRGESP) API is used by any program that needs an application programming interface to determine the encoding scheme, character set, and code page elements of a CCSID. This API will retrieve this information about the CCSID and then return the value of the encoding scheme associated with CCSID1 in ES, and the values of CS and CP elements in CSCPL.

The API makes the following checks on the input variables:

- The CCSID is verified. If the CCSID is an invalid value or out of range, an error will be sent.
- N1 and N2 are also verified for valid values. If an error is found in their values, the values will be reset and processing continued.

Required Parameter Group

CCSID1
INPUT; BINARY(31)
Variable containing the CCSID value passed by the invoker. Field type is 32-bit two's complement binary (a positive number from 1 to 65,534).

N1 INPUT; BINARY(31)
Variable containing the number of CS or CP elements in the output area (starting at CSCPL) for which space is allocated by the invoker. Field type is 32-bit two's complement binary (a positive value from 2 (space of 8 bytes in CSCPL) to 32 (space of 128 bytes in CSCPL) to accommodate up to 16 pairs of CS,CP values).

N1 must be an even number. If a value less than 2 is used, an error feedback code is returned. If an odd value is used, an error feedback code is sent.

N2 I/O; BINARY(31)
N2 cannot be greater than N1. If N2 is greater than N1, only N1 number of elements are placed in CSCPL.

This variable contains the number of values (each pair of CS/CP is counted as two values) associated with CCSID1 and returned in CSCPL providing sufficient space (N1) was allocated. Field type is a 32-bit two's complement binary.

At invocation, if the returned value of N2 is less than or equal to N1, all the available output data (N2 elements) are placed in the allocated space. If the returned value of N2 is greater than N1, there is more output data to be returned. N1 CS and CP values are placed in CSCPL. Any error detected in N1 and N2 is indicated in the feedback code FB.

When there is insufficient space allocated (N2), the overflow must be dealt with.

A common convention for this would be similar to:

Assume allocated space = 4, required space = 15, and N1 = allocated space, N2 = actually-used space.

```
Set N1 to 4
Set N2 to 0
Do until N1 >= N2
  Call CDRGESP (ccsid, N1, N2, ES, CSCPL, feedback)
  Process returned data
End-do
```

In this example, N2 is an input/output variable, and must be initialized to zero by the invoker. N2 is used to determine where to start the output data from. The invoker gets the data in pieces, starting from the beginning, using repeated calls to get the subsequent pieces, until all the remaining data fits into the allocated space.

ES OUTPUT; BINARY(31)
Variable contains the encoding scheme associated with CCSID1. Field type is 32-bit two's complement binary (a positive number in the range 4,096 to 65,535). Some encoding schemes are (in hexadecimal):

Figure 39-1. Encoding Scheme ID Values in CDRA Level 1

ESID in Hex	Encoding Scheme	Interpretation
X'0000 1100'	EBCDIC	Single byte, no code extension is allowed.
X'0000 2100'	IBM-PC Data	Single byte, no code extension is allowed.
X'0000 3100'	IBM-PC Display	Single byte, no code extension is allowed.
X'0000 4100'	ISO 8	Single byte, no code extension is allowed.
X'0000 5100'	ISO 7 (ASCII)	Single byte, no code extension is allowed.
X'0000 1200'	EBCDIC	Double byte, no code extension is allowed.
X'0000 2200'	IBM-PC Data	Double byte, no code extension is allowed.
X'0000 3200'	IBM-PC Display	Double byte, no code extension is allowed.
X'0000 1301'	EBCDIC	Mixed single byte and double byte, using SO/SI method.
X'0000 2300'	IBM-PC Data	Mixed single byte and double byte, no (explicit) code extension.
X'0000 3300'	IBM-PC Display	Mixed single byte and double byte, no (explicit) code extension.

Note: The first 2 bytes are reserved and must be hex 0s.

CSCPL
OUTPUT; ARRAY(N1)
Variable that is a N1-element array of pairs of 32-bit two's complement binary numbers in the format CS1, CP1...CS16, CP16. Each CS and CP is a positive number in the range 1-65,534 and is placed in a single CSCPL array element.

FB OUTPUT; ARRAY(3)
A feedback array containing the processing status (and any associated reason) returned by this function. Field type is an array of three 32-bit two's complement binary values (12 bytes or 96 bits). The status code is a positive number in the first 16 bits and the reason code is a positive number in the second 16 bits.

Feedback Codes

The feedback codes returned by this function are (in hex):

0000 0000	Successful
0001 0001	CCSID1 not found
0002 0001	CCSID1 is 0 - reserved number
0003 0001	CCSID1 is 65535 - reserved number
0004 0001	N1 length insufficient to contain returned output
0005 0002	N2 greater than N1 but inconsistent with number of elements returned
0007 0001	CCSID repository is invalid in structure
0008 0001	CCSID1 out of range
0008 0002	N1 is odd value
0008 0003	N1 is < 2

The following feedback codes are reserved:

- 0006 0001
- 0006 0002

Get Related Default CCSID (CDRGRDC) API

Parameters

Required Parameter Group:

1	CCSID1	Input	Binary(31)
2	ESIN	Input	Binary(31)
3	SEL	Input	Binary(31)
4	CCSIDR	Output	Binary(31)
5	FB	Output	Binary(31)

The Get Related Default CCSID (CDRGRDC) API is by any program that needs an application programming interface to allow the invoker to get a nearest equivalent or best fit[4] "related CCSID." The invoker supplied encoding scheme (ES) is used as an additional key to select the appropriate related CCSID.

This function is necessary when a given CCSID is not directly usable, such as in the following situations:

- CCSID is not locally supported

  A given CCSID is not one of the supported CCSIDs in that environment. For example, an AS/400 system supporting only EBCDIC CCSIDs may be serving a PC user, where all the data generated is in one of the PC CCSIDs. An AS/400 database server receiving the SQL statement `CREATE TABLE` from the PC user, may not be able to create a table in any other CCSIDs than an EBCDIC CCSID supported in that installation. Before the table creation is completed, a CCSID value is needed to identify the table's CCSID. Any data from the PC that is placed in this table will be converted to this CCSID. However, a selection from the locally supported CCSIDs of a single CCSID that can preserve the

[4] The system creates a list of CCSIDs (from those supported on the system) that have ES equal to ESIN. This list is used to find a CCSID value with a character set that is either equal to or is the best match available to CCSID1's character set. If found, that CCSID value is returned as CCSIDR; otherwise, a feedback code is returned to indicate a not found condition.

maximum number of PC graphic characters is needed. If the user does not supply this CCSID, the system has to default to a CCSID.

- CCSID to match a specific data type is needed

  In a situation where a given CCSID is incompatible with the data type (for example, SBCS CCSID and graphic data type), and a group of CCSIDs (one for each of SBCS, DBCS and Mixed SB/DB encoding schemes) are used, it is necessary to pick the correct CCSID that matches the data type.

  Most CCSIDs registered to date have only one CCSID per data type and ES match. Such CCSIDs share one or more CS and CP values among them, and differ only on the ES id values. However, since some CS and CP values can be shared between different CCSIDs (with the same ES), more than one CCSIDs can qualify to be used.

  For example, the CS, CP (00370, 00300) of DBCS Japanese Host CCSID 00300 is used in CCSID 05026 (with SBCS Katakana Extended CS 01172, CP 00290) and in CCSID 05035 (with SBCS Latin Extended CS 01172 and CP 01027). If only the DBCS CCSID 00300 was specified, both the CCSIDs 05026 and 05035 qualify to be a related mixed CCSID. Also, both the SBCS CCSIDs 00290 and 01027 qualify to be the related SBCS CCSID.

  However, a single default value selected from the multiple possible CCSIDs is predetermined and made available as a *related default* CCSID. This function gets this predetermined default for the invoker supplied CCSID value. The defaults are arranged with ES as the key to properly match the data type needed by the invoker.

Required Parameter Group

CCSID1
INPUT; BINARY(31)
Variable containing the CCSID value passed by the invoker. Field type is 32-bit two's complement binary (a positive number from 1 to 65,534).

The CCSID will be verified. If the CCSID is an invalid value or out of range, a feedback code will be returned to indicate an error.

ESIN
INPUT; BINARY(31)
Variable contains the Encoding Scheme value that is referenced. Field type is 32-bit two's complement binary (a zero, or a positive number in the range 4,352 – 65,534). For example formats, see Figure 39-1 on page 39-6.

Although this value is 4 bytes, it is the 2 least significant bytes that are used. ESIN will be verified to be within its specified range of valid values. If it is an invalid value, a feedback code will be returned to indicate an error, and CCSID1 will be copied and returned in CCSIDR.

SEL
INPUT; BINARY(31)
Variable that is reserved to identify any specific selection criteria as additional input, for example to select among two equally valid related defaults. Field type is 32-bit two's complement binary (a positive number). Zero is currently the only valid value, all others are reserved.

SEL will be verified to be within its specified range of valid values. If an invalid value is found, it is treated as 0.

CCSIDR
OUTPUT; BINARY(31)
Variable containing the CCSID value returned to the invoker. Field type is 32-bit two's complement binary (a positive number from 1 to 65,534). If no related default CCSID is defined for CCSID1, the CCSID1 value is copied and returned in CCSIDR.

FB OUTPUT; ARRAY(3)
A feedback array containing the processing status (and any associated reason) returned by this function. Field type is an array of three 32-bit two's complement binary values (12 bytes or 96 bits). The status code is a positive number in the first 16 bits and the reason code is a positive number in the second 16 bits.

Feedback Codes

The feedback codes in hexadecimal returned by this function are:

0000 0000	Successful
0001 0001	CCSID1 not found. CCSIDR = CCSID1
0002 0001	CCSID1 is 0 - reserved number
0003 0001	CCSID1 is 65535 - reserved number
0005 0001	SEL value was not zero
0007 0001	CCSID resource is invalid structure
0008 0001	CCSID1 out of range
0008 0002	ESIN out of range - CCSID2 copied to CCSIDR

These feedback codes are reserved:

0001 0002
0006 0001
0006 0002

Get Short Form CCSID (CDRSCSP) API

Parameters

Required Parameter Group:

1	CSCPL	Input	Array(32)
2	N1	Input	Binary(31)
3	ESIN	Input	Binary(31)
4	CCSIDR	Output	Binary(31)
5	ESR	Output	Binary(31)
6	FB	Output	Binary(31)

This Get Short Form CCSID (CDRSCSP) API is used by any program that needs an application programming interface

to determine the CCSID when the ES (encoding scheme) and CS,CP (character set,code page) values are known. The CS,CP are also known as the CGCSGID (coded graphic character set global identification). This API retrieves the CCSID information and returns it to the caller the CCSID value in the variable CCSIDR.

The API makes the following checks on the input variables:

- CS and CP values in CSCPL are verified to be within their specified ranges.
- N1 is verified to be within its specified range of valid values.

Required Parameter Group

CSCPL
INPUT; ARRAY(32)
Variable that is a 32-element array of 16 pairs of 32-bit two's complement binary numbers in the format CS1, CP1...CS16, CP16. Each CS and CP is a positive number in the range 1-65,534 and is placed in a single CSCPL array element.

N1 INPUT; BINARY(31)

Variable containing the number of CS or CP elements in the output area (starting at CSCPL) for which space is allocated by the invoker. Field type is 32-bit two's complement binary (a positive value from 2 (space of 8 bytes in CSCPL) to 32 (space of 128 bytes in CSCPL) to accommodate up to 16 pairs of CS,CP values).

N1 must be an even number. If a value less than 2 is used, an error feedback code is sent. If an odd value (within the valid range 2-32) is used, an error feedback code is sent.

Note: Currently, only 2 (1 CS element and 1 CP element) or 4 (1 CS element, 1 CP element and 1 CS element, 1 CP element) values are stored in the internal structure.

ESIN
INPUT; BINARY(31)

Variable contains the encoding scheme value that is referenced. Field type is 32-bit two's complement binary (a zero, or a positive number in the range 4,352 – 65,534). For example formats, see Figure 39-1 on page 39-6.

Although this value is 4 bytes, it is the 2 least significant bytes that are used.

A value of zero means that only one CCSID with the specified CS, CP is expected in the repository and it is to be returned.

Note: The current implementation of this API only supports a value of zero for this parameter.

CCSIDR
OUTPUT; BINARY(31)
Variable containing the CCSID value returned to the invoker. Field type is 32-bit two's complement binary (a positive number from 1 to 65,534). A value of 65,535 is returned when the function can not find the requested CCSID.

ESR
OUTPUT; BINARY(31)

Variable to contain the encoding scheme value of the returned CCSID. If multiple CCSIDs were found and ESIN was specified as zero, an error is returned in FB; the first CCSID value found and its associated ES value are returned. When a non-zero ESIN is specified, values of ESR and ESIN are equal. If the function finds only one CCSID and its ES value does not match the ESIN specified, the function returns the CCSID and its ES, with an error indication.

Note: The current implementation of this API does not support this function.

FB OUTPUT; BINARY(31)
A feedback array containing the processing status (and any associated reason) returned by this function. Field type is an array of three 32-bit two's complement binary values (12 bytes or 96 bits). The status code is a positive number in the first 16 bits and the reason code is a positive number in the second 16 bits.

Feedback Codes

The feedback codes returned by this function are:

0000 0000	Successful
0001 0001	CCSID not found
0001 0002	CCSID found for CSCPL value, but not with ESIN value
0002 0001	A CP element has an invalid value of 0
0002 0002	A CS element has an invalid value of 0
0003 0001	A CP element has an invalid value of 65535
0003 0002	A CS element has an invalid value of 65535
0007 0001	CCSID resource repository is invalid structure
0008 0001	A CS or CP value in CSCPL is outside of range
0008 0002	N1 is outside the range of 2-32
0008 0003	N1 is < 2
0008 0004	ESIN is outside the range. O will be used
0009 0001	More than one parameter in a function call points to same storage location

The following feedback codes are reserved:

- 0001 0003
- 0001 0004

Remove All Bookmarks from a Course (QEARMVBM) API

Parameters

Required Parameter Group:

1	Course ID	Input	Char(10)
2	Error code	I/O	Char(*)

The Remove All Bookmarks from a Course (QEARMVBM) API removes all bookmarks from a Tutorial System Support course. This API provides support similar to option 9 (Remove bookmarks) on the Work with Courses display within the Start Education (STREDU) command.

Authority

The user must be an education administrator and have one of the following authorities:

Authority *ALLOBJ or *SECADM

Required Parameter Group

Course ID
INPUT; CHAR(10)
The ID of the course that is to have all bookmarks removed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1D50 E Not authorized to remove bookmarks.
CPF1D51 E Not all bookmarks removed.
CPF1D52 E Course not found.
CPF3CF1 E Error code parameter not valid.

Retrieve Main Storage (QVTRMSTG) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Main storage address	Input	Char(12)
5	Main storage image selector	Input	Char(10)
6	Error code	I/O	Char(*)

The Retrieve Main Storage (QVTRMSTG) API retrieves information from main storage either from the current system or from the space where main storage was dumped from a previous initial program load (IPL).

You may want to use this API to write a tool that analyzes main storage and determines if the problem is a known problem and if a program temporary fix (PTF) exists for the problem.

Authority

Authority *SERVICE

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable to receive a copy of the contents of the main storage. This area must be large enough to accommodate the specified information.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. The length must be equal to or greater than 8 bytes. There is no limit to the variable length; however, if the length is not large enough to hold the copy of main storage, the data is truncated.

Format name
INPUT; CHAR(8)
The format of main storage being retrieved. Valid formats are:

STGI0100 Basic main storage information. See "STGI0100 Format."
STGI0200 Main storage information with tags. See "STGI0200 Format" on page 39-10.

Main storage address
INPUT; CHAR(12)
The address of main storage being retrieved (a character representation of a hexadecimal address). If format STGI0200 is specified, the address must be on a 16-byte boundary.

Main storage image selector
INPUT; CHAR(10)
The main storage image from which to retrieve information. You can use these special values:

***MSTOR** The current main storage of the system
***DMPSPC** The main storage dump space

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

STGI0100 Format

The following table shows the information returned in the receiving variable for the STGI0100 format. For a detailed description of each field, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(*)	Main storage returned

STGI0200 Format

The following table shows the information returned in the receiving variable for the STGI0200 format. For a detailed description of each field, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Main storage length
12	12	BINARY(4)	Tag length
16	16	CHAR(*)	Main storage returned
*	*	CHAR(*)	Tag information returned

Field Descriptions

Bytes available. The total length of data available. For the QVTRMSTG API, the length will be equal to the bytes returned.

Bytes returned. The length of the data actually returned. The number of bytes returned is always less than or equal to both the number of bytes available and the receiving variable length.

Main storage length. The length of the main storage returned.

Main storage returned. A copy of the main storage from the address requested. The data is returned in hexadecimal format; 1 byte of main storage is represented as 1 byte in the returned data.

Tag information returned. A copy of the tag information for the main storage returned. To determine where the tag information returned starts, add the tag length and main storage length, plus 16.

Tag length. The length of tag information returned.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C5B E Option template entry is not valid.
CPF6FE0 E Requested main storage address is not valid.
CPF6FE1 E Selected main store image must be either *MSTOR or *DMPSPC.
CPF6FE2 E Retrieved data is not complete.
CPF6FE3 E Error occurred during data retrieval.
CPF6FE4 E Address specified must be on a 16-byte boundary.

Part 20. Reference Information

Appendix A. Examples

This appendix contains example programs that use APIs. Using the example programs, you can:

- Delete old spooled files using a program in one of the following languages:
  - RPG
  - COBOL
  - Pascal
  - System C/400 PRPQ
- Change active jobs
- Change a job schedule entry
- Create your own telephone directory
- Create and manipulate a user index
- Create a batch machine
- Use the Create Program (QPRCRTPG) API
- Use profile handles
- Generate and send an alert
- Create diagnostic reports using message handling APIs
- List directories and subdirectories in spooled files
- Work with stream files
- Use the network management APIs
- Use the COBOL/400 APIs to control run units and error handling
- Use the user-defined communications APIs

Deleting Old Spooled Files

The following application program runs using the Spooled Information (SPOOLINFO) command. This example has three major parts:

1. The SPOOLINFO command calls the spool information (SPOOLINFO) program in one of the following languages:
   - RPG
   - COBOL
   - Pascal
   - System C/400 PRPQ
2. The SPOOLINFO program is supplied in RPG, COBOL, Pascal, or System C/400 PRPQ. It does the following before calling the application that deletes the spooled files and the user space (CLDLT program):
   a. Creates a user space (QUSCRTUS API).
   b. Generates a list of spooled files (QUSLSPL API).
   c. Retrieves information from a user space:
      - QUSRTVUS API
      - QUSPTRUS API
   d. Retrieve more spooled file attribute information received from the user space (QUSRSPLA API).
3. CL delete (CLDLT) program:
   a. Deletes the specified spooled files (DLTSPLF command).
   b. Sends a message if the spooled file was deleted (SNDPGMMSG command).
   c. Sends a message identifying how many spooled files were deleted (SNDPGMMSG command).
   d. Deletes the user space (DLTUSRSPC command).

SPOOLINFO Command Source

The command source for the SPOOLINFO command follows:

```
   /*                                                              */
   /* PROGRAM:  SPOOLINFO                                          */
   /*                                                              */
   /* LANGUAGE:  CL COMMAND SOURCE                                 */
   /*                                                              */
   /* DESCRIPTION:  COMMAND SOURCE FOR THE SPOOLINFO COMMAND,      */
   /*               WHICH CALLS THE SPOOLINFO PROGRAM.             */
   /*                                                              */
        CMD PROMPT('DELETE OLD SPOOLED FILES')
   /*  PARAMETERS FOR CREATING THE USER SPACE (QUSCRTUS)           */
        PARM KWD(USRSPC) TYPE(QUAL1) MIN(1) PROMPT('User Space:')
        PARM KWD(EXTATR) TYPE(*NAME) LEN(10) MIN(1) +
             PROMPT('Extended Attributes:')
        PARM KWD(SIZE) TYPE(*DEC) LEN(13 5) RANGE(1 16776704) +
             MIN(1) PROMPT('Size of User Space:')
        PARM KWD(INZ) TYPE(*CHAR) LEN(1) MIN(1) ALWUNPRT(*NO) +
             PROMPT('Initialize User Space To:')
        PARM KWD(AUTH) TYPE(*CHAR) LEN(10) MIN(1) RSTD(*YES) +
             VALUES(*CHANGE *ALL *USE *EXCLUDE) +
             PROMPT('Authority:')
        PARM KWD(TEXT) TYPE(*CHAR) LEN(50) MIN(1) ALWUNPRT(*NO) +
             PROMPT('Text Description:')
   /* PARAMETERS FOR LIST OF SPOOLED FILES (QUSLSPL)               */
        PARM KWD(SPLFMTNME) TYPE(*NAME) LEN(8) MIN(1) RSTD(*YES) +
             VALUES(SPLF0100) PROMPT('QUSLSPL Format Name:')
        PARM KWD(USRPRFNME) TYPE(*SNAME) LEN(10) MIN(1)    +
             SPCVAL(*ALL) PROMPT('User Profile Name:')
        PARM KWD(OUTQUEUE) TYPE(QUAL2) MIN(1) PROMPT('Output Queue:')
        PARM KWD(FORMTYPE) TYPE(*NAME) LEN(10) MIN(1) +
             SPCVAL(*ALL *STD) PROMPT('Form Type:')
        PARM KWD(USERDATA) TYPE(*CHAR) MIN(1) LEN(10) +
             SPCVAL(*ALL) PROMPT('User Data')
   /* PARAMETERS FOR SPOOLED FILE ATTRIBUTES (QUSRSPLA)            */
        PARM KWD(SPLAFMTNNE) TYPE(*NAME) MIN(1) RSTD(*YES) +
             LEN(10) VALUES(SPLA0100) PROMPT('QUSRSPLA Format Name:')
   /* INFORMATION NEEDED FOR PROGRAM                               */
        PARM KWD(DELETEDATE) TYPE(*DATE) +
             PROMPT('Last Deletion Date(MMDDYY):')
QUAL1:  QUAL TYPE(*NAME) LEN(10)
        QUAL TYPE(*NAME) LEN(10) SPCVAL(*CURLIB)  +
             PROMPT('Library Name:')
QUAL2:  QUAL TYPE(*NAME) LEN(10)  SPCVAL(*ALL)
        QUAL TYPE(*NAME) LEN(10) SPCVAL(*LIBL *CURLIB '          ') +
             PROMPT('Library Name:')
```

To create the CL command, specify the following:

```
CRTCMD CMD(QGPL/SPOOLINFO) PGM(QGPL/SPOOLINFO) +
    SRCFILE(QGPL/QCMDSRC) ALLOW(*IPGM *BPGM)
```

To delete old spooled files, you can use one of the application programs provided in the following languages:

- RPG
- COBOL
- Pascal
- System C/400 PRPQ

RPG SPOOLINFO Program: To delete old spooled files, use the following RPG program:

```
     H* ****************************************************************
     H* ****************************************************************
     H*                                                                *
     H* MODULE:  SPOOLINFO                                             *
     H*                                                                *
     H* LANGUAGE:  RPG                                                 *
     H*                                                                *
     H* FUNCTION:  THIS APPLICATION WILL DELETE OLD SPOOL FILES        *
```

```
H*           FROM THE SYSTEM, BASED ON THE INPUT PARAMETERS.    *
H*                                                              *
H* APIs USED:  QUSCRTUS, QUSLSPL, QUSRTVUS, QUSRSPLA            *
H*                                                              *
H* ************************************************************
H* ************************************************************
IUSRSPC      DS
I                                       1  10 USNAME
I                                      11  20 USLIB
ITGTDAT      DS
I                                       1   1 TGTCEN
I                                       2   3 TGTYR
I                                       4   5 TGTMTH
I                                       6   7 TGTDAY
IRCVVAR      DS
I                                    B  1   4 0OFFSET
I                                    B  9  12 0NOENTR
I                                    B 13  16 0LSTSIZ
IRCVAR1      DS
I                                       1  10 USRNM1
I                                      11  30 OUTQNA
I                                      31  40 USRDT1
I                                      41  50 FRMTY1
I                                      51  66 IJOBID
I                                      67  82 ISPLID
IRCVAR2      DS
I                                    B  1   4 0BYTRTN
I                                    B  5   8 0BYTVAL
I                                       9  24 JOBID
I                                      25  40 SPLFID
I                                      41  50 JOBNAM
I                                      51  60 USRNAM
I                                      61  66 JOBNUM
I                                      67  76 FILNAM
I                                    B  77  80 0FILNUM
I                                      81  90 FRMTYP
I                                      91 100 USRDTA
I                                     101 110 STATUS
I                                     111 120 FILAVL
I                                     121 130 HLDF
I                                     131 140 SAVF
I                                    B 141 144 0TOTPAG
I                                    B 145 148 0PAGWRT
I                                    B 149 152 0STRPAG
I                                    B 153 156 0ENDPAG
I                                    B 157 160 0LASPAG
I                                    B 161 164 0RESPRT
I                                    B 165 168 0TOTCPY
I                                    B 169 172 0PYLFT
I                                    B 173 176 0LPI
I                                    B 177 180 0CPI
I                                     181 182 OUTPRI
I                                     183 192 OUTQNM
I                                     193 202 OUTQLB
I                                     203 209 DATFOP
I                                     203 203 DATCEN
I                                     204 205 DATYR
I                                     206 207 DATMTH
I                                     208 209 DATDAY
I                                     210 215 TIMFOP
I                                     216 225 DEVFNA
I                                     226 235 DEVFLB
I                                     236 245 PGMOPF
I                                     246 255 PGMOPL
I                                     256 270 ACCCOD
I                                     271 300 PRTTXT
I                                    B 301 304 0RCDLEN
I                                    B 305 308 0MAXRCD
I                                     309 318 DEVCLS
I                                     319 328 PRTTYP
I                                     329 340 DOCNAM
I                                     341 404 FLDNAM
I                                     405 412 S36PRC
I                                     413 422 PRTFID
I                                     423 423 RPLUN
I                                     424 424 RPLCHR
I                                    B 425 428 0PAGLEN
I                                    B 429 432 0PAGWID
I                                    B 433 436 0NUMSEP
I                                    B 437 440 0OVRLIN
I                                     441 450 DBCSDA
I                                     451 460 DBCSEC
I                                     461 470 DBCSSO
I                                     471 480 DBCSCR
I                                    B 481 484 0DBCSCI
I                                     485 494 GRAPHI
I                                     495 504 CODPAG
I                                     505 514 FORNAM
I                                     515 524 FORLIB
I                                    B 525 528 0RCDRW
I                                     529 538 PRTFON
I                                     539 544 S36SPL
I                                    B 545 548 0PAGROT
I                                    B 549 552 0JUSTIF
I                                     553 562 PRTBOT
I                                     563 572 FLDRCD
I                                     573 582 CTLCHR
I                                     583 592 ALGFRM
I                                     593 602 PRTQUA
I                                     603 612 FRMFED
I                                     613 683 VOLUME
I                                     684 700 FLABID
I                                     701 710 EXCTYP
I                                     711 720 CHRCOD
I                                    B 721 724 0TOTRCD
I            DS
I                                    B  1   4 0LENDTA
I                                    B  5   8 0STRPOS
I                                    B  9  12 0SPLF#
I                                    B 13  16 0RCVLE1
I                                       17  22 FIL#
I                                    B 23  26 0RCVLE2
I            DS
I                                    B  1   4 0USSIZE
C* ************************************************************
C* ************************************************************
C*                                                              *
C*              EXECUTABLE CODE STARTS HERE                     *
C*                                                              *
C* ************************************************************
C* ************************************************************
C*                                                              *
C           *ENTRY    PLIST
C                     PARM           USRSPC 20
C                     PARM           EXTATR 10
C                     PARM           USSIZ  135
C                     PARM           USINIT  1
C                     PARM           USAUTH 10
C                     PARM           USTEXT 50
C                     PARM           FMTNME  8
C                     PARM           USRNME 10
C                     PARM           OUTQ   20
C                     PARM           FORTYP 10
C                     PARM           USRDAT 10
C                     PARM           FMTNM1  8
C                     PARM           DLTDAT  7
C                     Z-ADDUSSIZ     USSIZ1 155
C                     Z-ADDUSSIZ1    USSIZE
C                     DUMP
C                     MOVE DLTDAT    TGTDAT
C                     MOVE FORTYP    FRMTYP
C                     MOVE USRDAT    USRDTA
C                     Z-ADD0         DLTCNT 150
C*                                                              *
C* CREATE THE USER SPACE USING THE PARAMETERS FROM THE CL       *
C* COMMAND.                                                     *
C*                                                              *
C                     CALL 'QUSCRTUS'
C                     PARM           USRSPC
C                     PARM           EXTATR
C                     PARM           USSIZE
C                     PARM           USINIT
C                     PARM           USAUTH
C                     PARM           USTEXT
C*                                                              *
C* FILL THE USER SPACE JUST CREATED WITH SPOOL FILES AS         *
C* DEFINED IN THE CL COMMAND.                                   *
C*                                                              *
C                     CALL 'QUSLSPL'
C                     PARM           USRSPC
C                     PARM           FMTNME
C                     PARM           USRNME
C                     PARM           OUTQ
C                     PARM           FRMTYP
C                     PARM           USRDTA 10
C*                                                              *
```

```
C* ************************************************************
C* ************************************************************
C*                                                            *
C*     BEGINNING OF LOOP                                      *
C*                                                            *
C* ************************************************************
C* ************************************************************
C*                                                            *
C* USRSPC FILLED WITH SPOOL INFORMATION, NOW FIND OFFSET AND SIZE*
C* OF EACH ENTRY AND THE NUMBER OF ENTRIES. DO THIS USING THE  *
C* QUSRTVUS API.                                              *
C*                                                            *
C                     Z-ADD16        LENDTA
C                     Z-ADD125       STRPOS
C*                                                            *
C                     CALL 'QUSRTVUS'
C                     PARM           USRSPC
C                     PARM           STRPOS
C                     PARM           LENDTA
C                     PARM           RCVVAR
C*                                                            *
C* CHECK RCVVAR DATA STRUCTURE FOR NUMBER OF LIST ENTRIES, OFFSET*
C* TO LIST ENTRIES, AND SIZE OF EACH LIST ENTRY.              *
C* INFORMATION NEEDED FOR THE QUSLSPL API IS CONTAINED WITHIN *
C* THE FIRST 164 BYTES OF FORMAT SPLI0100.                    *
C* THIS WILL BE DONE FOR EACH ENTRY IN THE USER SPACE.        *
C*                                                            *
C                     Z-ADDOFFSET    STRPOS
C                     ADD  1         STRPOS
C                     Z-ADDLSTSIZ    LENDTA
C                     Z-ADD164       RCVLE1                CVD
C                     Z-ADD209       RCVLE2                CVD
C                     Z-ADD1         COUNT  150
C           COUNT     DOWLENOENTR
C*                                                            *
C* RETRIEVE THE INFORMATION FROM THE USER SPACE ABOUT THE SPOOL*
C* FILE.                                                      *
C*                                                            *
C                     CALL 'QUSRTVUS'
C                     PARM           USRSPC
C                     PARM           STRPOS
C                     PARM           LENDTA
C                     PARM           RCVAR1
C*                                                            *
C* NOW RETRIEVE SPOOL ATTRIBUTES USING THE INFORMATION IN THE *
C* USER SPACE, WHICH WAS RETRIEVED BEFORE THIS COMMENT.       *
C*                                                            *
C                     MOVE IJOBID    JOBID
C                     MOVE ISPLID    SPLFID
C                     MOVE *BLANKS   JOBINF
C                     MOVEL'*INT'    SPLFNM 10
C                     MOVE *BLANKS   SPLF#
C                     MOVEL'*INT'    JOBINF 26
C*                                                            *
C                     CALL 'QUSRSPLA'
C                     PARM           RCVAR2
C                     PARM           RCVLE2
C                     PARM           FMTNM1
C                     PARM           JOBINF
C                     PARM           JOBID
C                     PARM           SPLFID
C                     PARM           SPLFNM
C                     PARM           SPLF#
C*                                                            *
C* CHECK RCVAR1 DATA STRUCTURE FOR DATE FILE OPENED.          *
C*                                                            *
C* DELETE SPOOL FILES THAT ARE OLDER THAN THE TARGET DATE     *
C* SPECIFIED ON THE COMMAND.  A MESSAGE IS SENT FOR EACH SPOOL *
C* FILE DELETED.                                              *
C*                                                            *
C*                                                            *
C                     MOVE '1'       OPTION  1
C           DATYR     IFLT TGTYR
C                     EXSR CLDLT
C                     ELSE
C           DATYR     IFEQ TGTYR
C           DATMTH    IFLT TGTMTH
C                     EXSR CLDLT
C                     ELSE                              NOT LT MTH
C           DATMTH    IFEQ TGTMTH
C           DATDAY    IFLE TGTDAY
C                     EXSR CLDLT
C                     END                               FOR LE DAY
C                     END                               FOR EQ MTH
C                     END                               FOR ELSE MTH
C                     END                               FOR EQ YR
C                     END                               FOR ELSE YR
C*                                                            *
C* GO  BACK AND PROCESS THE REST OF THE ENTRIES IN THE USER   *
C* SPACE.                                                     *
C           LSTSIZ    ADD  STRPOS    STRPOS
C           1         ADD  COUNT     COUNT
C                     END
C* ************************************************************ *
C* ************************************************************ *
C*                                                            *
C*            END OF LOOP                                     *
C*                                                            *
C* ************************************************************ *
C* ************************************************************ *
C*                                                            *
C* AFTER ALL SPOOL FILES HAVE BEEN DELETED THAT MEET OUR      *
C* REQUIREMENTS, SEND A FINAL MESSAGE TO THE USER.            *
C* DELETE THE USER SPACE OBJECT THAT WAS CREATED.             *
C*                                                            *
C                     MOVE '2'       OPTION
C                     EXSR CLDLT
C*                                                            *
C*                                                            *
C* ************************************************************ *
C* ************************************************************ *
C*                                                            *
C*                      END OF PROGRAM                        *
C*                                                            *
C* ************************************************************ *
C                     RETRN
C*
C* ************************************************************ *
C*                                                            *
C*                  CLDLT  SUBROUTINE                         *
C*                                                            *
C* THIS SUBROUTINE CALLS A CL PROGRAM THAT WILL EITHER:       *
C* -- DELETE A SPOOL FILE AND SEND A MESSAGE(OPTION = 1)      *
C* -- SEND A MESSAGE AND DELETE A USER SPACE OBJECT(OPTION = 2) *
C*                                                            *
C* ************************************************************ *
C*                                                            *
C           CLDLT     BEGSR
C*                                                            *
C* KEEP A COUNTER OF HOW MANY SPOOL FILES ARE DELETED.        *
C*                                                            *
C           OPTION    IFEQ '1'
C                     ADD  1         DLTCNT
C                     END
C                     MOVE FILNUM    FIL#
C                     CALL 'CLDLT'
C                     PARM           OPTION
C                     PARM           USNAME
C                     PARM           USLIB
C                     PARM           FILNAM
C                     PARM           JOBNUM
C                     PARM           USRNAM
C                     PARM           JOBNAM
C                     PARM           FIL#
C                     PARM           FRMTYP
C                     PARM           USRDTA
C                     PARM           DLTCNT
C                     ENDSR
```

To create the RPG program, specify the following:

```
CRTRPGPGM PGM(QGPL/SPOOLINFO) SRCFILE(QGPL/QRPGSRC)
```

COBOL SPOOLINFO Program: To delete spooled files, you can use this COBOL SPOOLINFO program:

```
      ************************************************************
      *                                                          *
      * PROGRAM:  DLTSPLF                                        *
      *                                                          *
      * LANGUAGE:  COBOL                                         *
      *                                                          *
      * DESCRIPTION:  DELETE OLD SPOOL FILES                     *
      *                                                          *
      * APIs USED:  QUSCRTUS, QUSLSPL, QUSRTVUS, QUSRSPLA        *
      *                                                          *
```

```
*****************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  DLTSPLF.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. IBM-AS400.
 OBJECT-COMPUTER. IBM-AS400.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

 DATA DIVISION.
 FILE SECTION.

 WORKING-STORAGE SECTION.
*****************************************************************
* VALUES USED FOR THE QUSRTVUS PROGRAM                          *
*****************************************************************
 01  RTV-START-POS PIC S9(9) BINARY VALUE 1.
 01  RTV-LENGTH    PIC S9(9) BINARY VALUE 140.
 01  RTV-HDR-INFO.
     05  FILLER PIC X(124).
     05  RTV-OFFSET-TO-ENTRIES PIC S9(9) BINARY.
     05  FILLER PIC X(4).
     05  RTV-NUMBER-OF-ENTRIES PIC S9(9) BINARY.
     05  RTV-SIZE-OF-EACH-ENTRY PIC S9(9) BINARY.

*****************************************************************
* VALUES USED FOR THE QUSRTVUS PROGRAM                          *
*****************************************************************
 01  LSTSPLA-INFO.
     05  LSTSPL-USER-NAME   PIC X(10).
     05  LSTSPL-OUTQ-NAME   PIC X(10).
     05  LSTSPL-OUTQLIB     PIC X(10).
     05  LSTSPL-USERDATA    PIC X(10).
     05  LSTSPL-FORMTYPE    PIC X(10).
     05  LSTSPL-INT-JOB-ID  PIC X(16).
     05  LSTSPL-INT-SPOOLID PIC X(16).

 01  RTVSPLA-JOB-ID         PIC X(26) VALUE '*INT'.

*****************************************************************
* VALUES USED FOR THE QUSRTVUS PROGRAM                          *
*****************************************************************
 01  RSPLA-DATE.
     05 R-CENTURY PIC X.
     05 R-MONTH   PIC X(2).
     05 R-DAY     PIC X(2).
     05 R-YEAR    PIC X(2).
 01  RSPLA-JOB-NAME PIC X(26) VALUE '*INT'.
 01  RSPLA-NAME PIC X(10) VALUE '*INT'.
 01  RSPLA-NUMBER PIC S9(9) BINARY VALUE -1.
 01  SPLA-VAR-LENGTH PIC S9(9) BINARY VALUE 800.
 01  SPLA-VAR.
     05 FILLER PIC X(40).
     05 SPLA-FILENAME PIC X(10).
     05 SPLA-JOBNAME  PIC X(10).
     05 SPLA-USERNAME PIC X(10).
     05 SPLA-JOBNUMBER PIC X(6).
     05 SPLA-FILENUMBER PIC S9(9) BINARY.
     05 SPLA-USERDATA PIC X(10).
     05 SPLA-FORMTYPE PIC X(10).
     05 FILLER PIC X(102).
     05 OPENED-DATE PIC X(7).
     05 FILLER PIC X(102).
     05 FILLER PIC X(593).
 01  DLT-OPTION PIC X VALUE '1'.
 01  DLT-COUNT PIC 9(15) PACKED-DECIMAL VALUE 0.
 01  DLT-SPL-NUMBER PIC 9(6).

*****************************************************************
* PARAMETERS THAT ARE PASSED TO THIS PROGRAM FROM THE COMMAND   *
*****************************************************************
 LINKAGE SECTION.
 01 PARM-SPCNAME.
     05 PARM-SPACE PIC X(10).
     05 PARM-LIB   PIC X(10).
 01 PARM-EXTATTR PIC X(10).
 01 PARM-SPCSIZE PIC 9(8) PACKED-DECIMAL.
 01 PARM-INITSPACE PIC X.
 01 PARM-AUTHORITY PIC X(10).
 01 PARM-DESCRIPTION PIC X(50).
 01 PARM-LSPL-FORMAT PIC X(8).
 01 PARM-USERNAME PIC X(10).
 01 PARM-OUTQ PIC X(20).
 01 PARM-FORMTYPE PIC X(10).
 01 PARM-USERDATA PIC X(10).
 01 PARM-RSPLA-FORMAT PIC X(10).
 01 PARM-DATE.
     05 P-CENTURY PIC X.
     05 P-MONTH   PIC X(2).
     05 P-DAY     PIC X(2).
     05 P-YEAR    PIC X(2).

*****************************************************************
* BEGINNING OF EXECUTABLE CODE.                                 *
*****************************************************************

 PROCEDURE DIVISION USING PARM-SPCNAME,
                          PARM-EXTATTR,
                          PARM-SPCSIZE,
                          PARM-INITSPACE,
                          PARM-AUTHORITY,
                          PARM-DESCRIPTION,
                          PARM-LSPL-FORMAT,
                          PARM-USERNAME,
                          PARM-OUTQ,
                          PARM-FORMTYPE,
                          PARM-USERDATA,
                          PARM-RSPLA-FORMAT,
                          PARM-DATE.

 MAIN-PROGRAM.

*    ***********************************************************
*    * CREATE THE USER SPACE USING INPUT PARMS FOR THE CALL   *
*    ***********************************************************

     CALL "QUSCRTUS" USING PARM-SPCNAME,
                           PARM-EXTATTR,
                           PARM-SPCSIZE,
                           PARM-INITSPACE,
                           PARM-AUTHORITY,
                           PARM-DESCRIPTION.

*    ***********************************************************
*    * LIST THE SPOOL FILES TO THE USER SPACE OBJECT.         *
*    ***********************************************************

     CALL "QUSLSPL"  USING PARM-SPCNAME,
                           PARM-LSPL-FORMAT,
                           PARM-USERNAME,
                           PARM-OUTQ,
                           PARM-FORMTYPE,
                           PARM-USERDATA.

*    ***********************************************************
*    * RETRIEVE ENTRY INFORMATION FROM THE USER SPACE.        *
*    ***********************************************************

     CALL "QUSRTVUS" USING PARM-SPCNAME,
                           RTV-START-POS,
                           RTV-LENGTH,
                           RTV-HDR-INFO.

*    ***********************************************************
*    * IF ANY SPOOL FILES WERE FOUND MATCHING THE SEARCH      *
*    * CRITERIA RETRIEVE DETAILED INFORMATION AND DECIDE      *
*    * WHETHER TO DELETE THE FILE OR NOT.                     *
*    ***********************************************************

     IF RTV-NUMBER-OF-ENTRIES IS GREATER THAN ZERO THEN
         ADD 1 TO RTV-OFFSET-TO-ENTRIES GIVING RTV-START-POS
         PERFORM CHECK-AND-DELETE THROUGH
             CHECK-AND-DELETE-END RTV-NUMBER-OF-ENTRIES TIMES.

*    ***********************************************************
*    * CALL THE CL PROGRAM "CLDLT" TO DELETE THE USER SPACE   *
*    * WE CREATED, AND TO SEND A MESSAGE TELLING HOW MANY     *
*    * SPOOL FILES WERE DELETED.                              *
*    ***********************************************************
     MOVE 'X' TO DLT-OPTION.

     CALL "CLDLT" USING DLT-OPTION,
                        PARM-SPACE,
                        PARM-LIB,
                        SPLA-FILENAME,
```

```
                          SPLA-JOBNUMBER,
                          SPLA-USERNAME,
                          SPLA-JOBNAME,
                          SPLA-FILENUMBER,
                          SPLA-FORMTYPE,
                          SPLA-USERDATA,
                          DLT-COUNT.

           STOP RUN.

       CHECK-AND-DELETE.

           CALL "QUSRTVUS" USING PARM-SPCNAME,
                                 RTV-START-POS,
                                 RTV-SIZE-OF-EACH-ENTRY,
                                 LSTSPLA-INFO.

           ADD RTV-SIZE-OF-EACH-ENTRY TO RTV-START-POS GIVING
               RTV-START-POS.

           CALL "QUSRSPLA" USING SPLA-VAR,
                                 SPLA-VAR-LENGTH,
                                 PARM-RSPLA-FORMAT,
                                 RSPLA-JOB-NAME,
                                 LSTSPL-INT-JOB-ID,
                                 LSTSPL-INT-SPOOLID,
                                 RSPLA-NAME,
                                 RSPLA-NUMBER.

           IF R-YEAR IS LESS THAN P-YEAR THEN
             PERFORM DLT-SPLF THROUGH DLT-SPLF-END
           ELSE
             IF R-YEAR IS EQUAL TO P-YEAR THEN
               IF R-MONTH IS LESS THAN P-MONTH THEN
                 PERFORM DLT-SPLF THROUGH DLT-SPLF-END
               ELSE
                 IF R-MONTH IS EQUAL TO P-MONTH THEN
                   IF R-DAY IS LESS THAN OR EQUAL TO P-DAY THEN
                     PERFORM DLT-SPLF THROUGH DLT-SPLF-END.

       CHECK-AND-DELETE-END.

       DLT-SPLF.

           ADD 1 TO DLT-COUNT.
           MOVE SPLA-FILENUMBER TO DLT-SPL-NUMBER.

           CALL "CLDLT" USING DLT-OPTION,
                              PARM-SPACE,
                              PARM-LIB,
                              SPLA-FILENAME,
                              SPLA-JOBNUMBER,
                              SPLA-USERNAME,
                              SPLA-JOBNAME,
                              DLT-SPL-NUMBER,
                              SPLA-FORMTYPE,
                              SPLA-USERDATA,
                              DLT-COUNT.

       DLT-SPLF-END.
```

To create the COBOL program, specify the following:

```
CRTCBLPGM PGM(QGPL/SPOOLINFO) SRCFILE(QGPL/QCBLSRC)
```

Pascal SPOOLINFO Program: To delete spooled files, you can use this Pascal SPOOLINFO program:

```
(*****************************************************************)
(* PROGRAM:  SPOOLINFO                                           *)
(*                                                               *)
(* LANGUAGE:  PASCAL                                             *)
(*                                                               *)
(* DESCRIPTION:  THIS IS AN EXAMPLE PROGRAM FOR THE USE OF USER  *)
(*               SPACES WRITTEN IN AS/400 PASCAL.  THIS PROGRAM  *)
(*               WILL CREATE A USER SPACE AND LIST THE SPOOLED   *)
(*               FILES REQUESTED IN THE LIST ENTRY SECTION.  SOME*)
(*               DETAILED INFORMATION ABOUT THE SPOOLED FILE IS  *)
(*               RETRIEVED.  IF THE CREATION DATE OF THE FILE IS *)
(*               OLDER THAN THE COMPARISON DATE, THEN THE FILE IS*)
(*               DELETED.  AFTER ALL THE OLD SPOOLED FILES ARE   *)
(*               DELETED, THE USER SPACE IS DELETED.             *)
(*                                                               *)
(* APIs USED:  QUCRTUS, QUSPTRUS, QUSLSPL, QUSRSPLA              *)
(*                                                               *)
(*****************************************************************)
program spoolinfo;

type
  sysname        = packed array (.1..10.) of char;
  qname          = packed array (.1..20.) of char;
  format         = packed array (.1..8.) of char;
  int_id         = packed array (.1..16.) of char;
  num_type       = packed array (.1..6.) of char;
  text_type      = packed array (.1..50.) of char;
  date_type      = packed array (.1..7.) of char;
  qjob_name_type = packed array (.1..26.) of char;
  rcv_var_type   = packed array (.1..724.) of char;
  count_type     = packed array (.1..8.) of char;
  size_type      = packed array (.1..7.) of char;
  header_space   = space (.400.) of integer;
  entry_space    = space (.96.) of int_id;
  error_code     = record
                   bytes_prov   : integer;
                   bytes_avail  : integer;
                   except_id    : size_type;
                   reserved     : string(1);
                   except_data  : text_type
                   end;

  text_format    = record
                   msg_text     : string(30);
                   api_text     : string(10);
                   end;
var
(* variables passed into this program                          *)
  us_name    :qname;     (* qualified userspace name           *)
  us_ext_attr:sysname;   (* extended attributes of userspace   *)
  us_size_s  :size_type; (* size of userspace                  *)
  us_init_val:char;      (* initial value of userspace         *)
  us_auth    :sysname;   (* authority of userspace             *)
  us_text    :text_type; (* text for userspace                 *)
  lspl_fmt   :format;    (* format of list entries from QUSLSPL*)
  user_name  :sysname;   (* user to delete spooled files from  *)
  outq_name  :qname;     (* output queue to delete files from  *)
  form_type  :sysname;   (* form type of spooled files to delete*)
  user_data  :sysname;   (* user data of spooled files to delete*)
  rspla_fmt  :format;    (* format of info returned from QUSRSPLA*)
  dlt_date   :date_type; (* cut-off date for deleted spooled files*)

(* variables passed as parameters for QUSCRTUS program         *)
(* most were declared above as input parameters                *)
  us_size_i  :integer;   (* integer value of userspace size    *)
  temp       :STRING(16);
  replace    :sysname;
(* variables passed as parameters for QUSPTRUS program         *)
  us_header_ptr  :@header_space; (* pointer to the userspace   *)

(* variables passed as parameters for QUSLSPL program          *)
(* these are all declared above as input parameters            *)

(* variables passed as parameters for QUSRSPLA program         *)
  rcv_var    :rcv_var_type;    (* returned information        *)
  rcv_var_len:integer;         (* length of returned variable *)
  qjob_name  :qjob_name_type;  (* qualified job name, set to *INT*)
  int_job_id :int_id;          (* internal job identifier     *)
  int_spl_id :int_id;          (* internal spooled file identifier*)
  splf_name  :sysname;         (* name of spooled file        *)
  splf_number:integer;         (* number of spooled file      *)

(* variables passes as parameters for CLDLT program *)
  option     :char;            (* delete option, 1=splf 2=usrspc  *)
  d_us_name  :sysname;         (* userspace to delete if option=2 *)
  d_us_lib   :sysname;         (* userspace library               *)
  d_splf     :sysname;         (* spl file to delete if option=1  *)
  d_job_num  :num_type;        (* job number for spooled file     *)
  d_user_name:sysname;         (* user name for spooled file      *)
  d_job_name :sysname;         (* job name for spooled file       *)
  d_splf_num :num_type;        (* number of spooled file to delete*)
  d_form_type:sysname;         (* spooled file form type          *)
  d_user_data:sysname;         (* spooled file user data          *)
  dlt_count  :count_type;      (* count of deleted spooled files  *)

(* declare pointers to receive input parameters since PARM     *)
(* function returns type POINTER                               *)
  qname_ptr  :@qname;
```

Examples: Deleting Old Spooled Files

```
  sysname_ptr:@sysname;
  s8_ptr     :@string(8);
  char_ptr   :@char;
  text_ptr   :@text_type;
  format_ptr :@format;
  date_ptr   :@date_type;
  size_ptr   :@size_type;

(* information retrieved from userspace header after QUSLSPL       *)
  list_offset     :integer;    (* decimal offset to list beginning *)
  num_of_entries  :integer;    (* number of entries in list        *)
  size_of_entries :integer;    (* size of each entry in list       *)
  list_entry_ptr  :@entry_space; (* pointer to a list entry        *)

(* some other variables used in the program                        *)
  count           :integer;    (* loop counter                     *)
  dlt_count_i     :integer;    (* integer value for delete count   *)
  rtv_date        :date_type;  (* date of retrieved spooled file   *)
  i_ptr           :@integer;   (* used for a conversion routine    *)
  msg_id          :size_type;
  msg_file        :qname;
  msg_length      :integer;
  msg_type        :sysname;
  msg_queue       :sysname;
  stack_cnt       :integer;
  msg_key         :num_type;
  err_code        :error_code;
  api_code        :integer;
  msg_data        :text_format;

(* declare external procedures                                     *)
(* create a userspace                                              *)
procedure QUSCRTUS (VAR us_name    :qname;
                    VAR ext_attr   :sysname;
                    VAR init_size  :integer;
                    VAR init_val   :char;
                    VAR us_auth    :sysname;
                    VAR text       :text_type;
                    VAR replace    :sysname;
                    VAR err_code   :error_code);nonpascal;

(* get a pointer to a userspace                                    *)
procedure QUSPTRUS (VAR us_name  :qname;
                    VAR us_ptr   :POINTER);nonpascal;

(* list spooled files in a userspace                               *)
procedure QUSLSPL (VAR us_name   :qname;
                   VAR fmt       :format;
                   VAR usr       :sysname;
                   VAR outq      :qname;
                   VAR form      :sysname;
                   VAR usr_data  :sysname;
                   VAR err_code  :error_code);nonpascal;

(* retrieve detailed information of a spooled file                 *)
procedure QUSRSPLA (VAR rcv_var    :rcv_var_type;
                    VAR rcv_len    :integer;
                    VAR fmt        :format;
                    VAR q_job_name :qjob_name_type;
                    VAR int_job_id :int_id;
                    VAR int_spl_id :int_id;
                    VAR splf_name  :sysname;
                    VAR splf_num   :integer;
                    VAR err_code   :error_code);nonpascal;

(* delete a userspace or a spooled file according to option        *)
procedure CLDLT (VAR option    :char;
                 VAR us_name   :sysname;
                 VAR us_lib    :sysname;
                 VAR splf      :sysname;
                 VAR user_name :sysname;
                 VAR job_num   :num_type;
                 VAR job_name  :sysname;
                 VAR splf_num  :num_type;
                 VAR fm_type   :sysname;
                 VAR usr_data  :sysname;
                 VAR dlt_cnt   :count_type);nonpascal;

procedure QMHSNDPM (VAR msg_id     :size_type;
                    VAR msg_file   :qname;
                    VAR msg_data   :text_format;
                    VAR msg_length :integer;
                    VAR msg_type   :sysname;
                    VAR msg_queue  :sysname;
                    VAR stack_cnt  :integer;
                    VAR msg_key    :num_type;
                    VAR err_code   :error_code);nonpascal;

procedure error_check;
   begin
      if err_code.bytes_avail <> 0 then
         begin
         msg_id      := 'CPF9898';
         msg_file    := 'QCPFMSG   *LIBL';
         msg_data.msg_text := 'An error has occured calling ';
         case api_code of
            1 : msg_data.api_text := 'QUSCRTUS. ';
            2 : msg_data.api_text := 'QUSPTRUS. ';
            3 : msg_data.api_text := 'QUSLSPL.  ';
            4 : msg_data.api_text := 'QUSRSPLA. ';
         end;
         msg_length := 40;
         msg_type   := '*ESCAPE';
         msg_queue  := '*';
         stack_cnt  := 1 ;
         QMHSNDPM (msg_id,
                   msg_file,
                   msg_data,
                   msg_length,
                   msg_type,
                   msg_queue,
                   stack_cnt,
                   msg_key,
                   err_code)
      end
      else
      return;
   end;

begin
(* get parameters passed into this program using PARM function     *)
qname_ptr  :=PARM(1);
us_name    :=qname_ptr@;   (* qualified name of userspace          *)
sysname_ptr:=PARM(2);
us_ext_attr:=sysname_ptr@; (* extended attributes of userspace     *)
size_ptr   :=PARM(3);
us_size_s  :=size_ptr@;    (* size of userspace in packed decimal  *)
char_ptr   :=PARM(4);
us_init_val:=char_ptr@;    (* initial value of userspace           *)
sysname_ptr:=PARM(5);
us_auth    :=sysname_ptr@; (* authority of userspace               *)
text_ptr   :=PARM(6);
us_text    :=text_ptr@;    (* text for userspace                   *)
format_ptr :=PARM(7);
lspl_fmt   :=format_ptr@;  (* format parameter for QUSLSPL program *)
sysname_ptr:=PARM(8);
user_name  :=sysname_ptr@; (* user name to delete spooled files    *)
qname_ptr  :=PARM(9);
outq_name  :=qname_ptr@;   (* out queue to delete spooled files    *)
sysname_ptr:=PARM(10);
form_type  :=sysname_ptr@; (* form type of delete spooled files    *)
sysname_ptr:=PARM(11);
user_data  :=sysname_ptr@; (* user data of delete spooled files    *)
format_ptr :=PARM(12);
rspla_fmt  :=format_ptr@; (* format parameter for QUSRSPLA program *)
date_ptr   :=PARM(13);
dlt_date   :=date_ptr@;   (* delete any files older than this date *)

(* call QUSCRTUS to create the user space                          *)
temp := str(us_size_s);
us_size_i := ptoi(str(temp),13,5); (* convert the size to int *)
replace := '*NO       ';
err_code.bytes_prov := 16;
api_code := 1;
QUSCRTUS(us_name,
         us_ext_attr,
         us_size_i,
         us_init_val,
         us_auth,
         us_text,
         replace,
         err_code);

error_check;
```

```
(* call QUSPTRUS to get a pointer to the user space header section *)
api_code := 2;
QUSPTRUS(us_name,us_header_ptr);

(* call QUSLSPL to list the spooled files in the user space       *)
api_code := 3;
QUSLSPL(us_name,
        lspl_fmt,
        user_name,
        outq_name,
        form_type,
        user_data,
        err_code);
error_check;

(* get info from header section of user space                     *)
list_offset     := us_header_ptr@(.124.);
num_of_entries  := us_header_ptr@(.132.);
size_of_entries := us_header_ptr@(.136.);

(* set pointers to beginning of the list entry section            *)
list_entry_ptr  := ptrto(us_header_ptr@(.list_offset.));

(* set up some variable for delete spooled file loop              *)
rcv_var_len     := 724;       (* size of return variable         *)
qjob_name       := '*INT                    ';(* use internal id *)
splf_name       := '*INT      ';              (* use internal id *)
splf_number     := -1;        (* this will be ignored            *)
option          := '1';       (* set option to delete spooled files*)
d_us_name       := substr(str(us_name),1,10);  (* parse qualified  *)
d_us_lib        := substr(str(us_name),11,10); (* userspace name   *)
dlt_count_i     := 0;         (* initialize delete count to zero  *)

(* loop through list entry section and retrieve spooled file      *)
(* attributes, check the date, and delete if too old              *)
for count := 1 to num_of_entries do    (* check all list entries  *)
  begin
  int_job_id := list_entry_ptr@(.50.); (* get internal job id      *)
  int_spl_id := list_entry_ptr@(.66.); (* get internal splf id     *)
(* retrieve the detailed spooled file information                 *)
api_code := 4;
  QUSRSPLA(rcv_var,
           rcv_var_len,
           rspla_fmt,
           qjob_name,
           int_job_id,
           int_spl_id,
           splf_name,
           splf_number,
           err_code);
error_check;

  rtv_date := substr(str(rcv_var),203,7);     (* get date of file  *)

(* delete the spooled file if too old                             *)
  if rtv_date <= dlt_date then        (* check date               *)
    begin
    dlt_count_i:= dlt_count_i + 1;    (* increment delete count   *)
    d_splf     := substr(str(rcv_var),67,10); (* get splf name     *)
    d_user_name:= substr(str(rcv_var),51,10); (* get user name     *)
    d_job_num  := substr(str(rcv_var),61,6);  (* get job number    *)
    d_job_name := substr(str(rcv_var),41,10); (* get job name      *)
    i_ptr      := ptrto(rcv_var(.77.));       (* get integer       *)
    d_splf_num := itoz(i_ptr@,6,0);           (* convert to zoned  *)
    d_form_type:= substr(str(rcv_var),81,10); (* get form type     *)
    d_user_data:= substr(str(rcv_var),91,10); (* get user data     *)
    dlt_count  := itop(dlt_count_i,15,0);     (* convert to packed *)
(* delete the spooled file                                        *)
    CLDLT(option,
          d_us_name,
          d_us_lib,
          d_splf,
          d_user_name,
          d_job_num,
          d_job_name,
          d_splf_num,
          d_form_type,
          d_user_data,
          dlt_count);
  end(*if*);

(* set list entry pointer to the next entry in the list           *)
  list_entry_ptr  := ptrto(list_entry_ptr@(.82.));
end(*for*);

(* delete the userspace that was created for the spooled file list *)
option := '2';            (* set option to two to delete userspace *)
dlt_count  := itop(dlt_count_i,15,0);         (* convert to packed *)
CLDLT(option,
      d_us_name,
      d_us_lib,
      d_splf,
      d_user_name,
      d_job_num,
      d_job_name,
      d_splf_num,
      d_form_type,
      d_user_data,
      dlt_count);
end.
```

To create a Pascal program, specify the following:

```
CRTPASPGM PGM(QGPL/SPOOLINFO) SRCFILE(QGPL/QPASSRC)
```

System C/400 PRPQ SPOOLINFO Program: To delete spooled files, you can use this System C/400 PRPQ SPOOLINFO program:

```
/*****************************************************************/
/*  PROGRAM:  SPOOLINFO                                          */
/*  DESCRIPTION:  THIS IS AN EXAMPLE PROGRAM FOR THE USE OF      */
/*                USER SPACES WRITTEN IN SYSTEM C/400.  THE FLOW */
/*                OF THIS PROGRAM IS AS FOLLOWS:                 */
/*               (1) MAKE A USER SPACE USING QUSCRTUS            */
/*               (2) INSERT SPOOLED FILES INTO SPACE USING QUSLSPL */
/*               (3) KEEP POINTER TO ENTRY LIST IN THE USER SPACE */
/*               (4) ENTER LOOP                                  */
/*                     RETRIEVE LIST ENTRY                       */
/*                     RETRIEVE MORE INFORMATION USING QUSRSPLA  */
/*                     IF SPOOLED FILE IS TOO OLD                */
/*                        DELETE SPOOLED FILE                    */
/*                        INCREMENT DELETE COUNTER               */
/*                    END LOOP                                   */
/*               (5) DELETE USER SPACE                           */
/*                                                               */
/*  APIs USED:  QUSCRTUS, QUSLSPL, QUSRSPLA, QUSPTRUS            */
/*                                                               */
/*****************************************************************/
#include "OPUSAPI.h"    /*Linkage info, structures         */
#include "opmhapi.h"
#include <string.h>     /*strcpy, strncpy, strcmp          */
#include <micomput.h>   /* _DPA_Template_T, cvtnci, cvtcni */


#pragma linkage(CLDLT,OS)
  void CLDLT (char option_num[1],
              char space_name[10],
              char space_lib[10],
              char file_name[10],
              char job_number[6],
              char usr_name[10],
              char job_name[10],
              char file_number[6],
              char form_type[10],
              char usr_data[10],
              char delete_count[15]);

  void error_check (void);

header_struct *space;
char *list_section;
splf0100 *entry_list;
spla0100 *Rcv_Spl_Var;
_DPA_Template_T d_tmp_t,k_tmp_t;
/*****************************************************************/
/* PARMS FOR CLDLT                                               */
/*****************************************************************/
char dlt_splf[] = "1";
char dlt_usrspc[] = "2";
char dlt_spc_name[10];
char dlt_spc_lib[10];
char job_nmbr[6];
char usr_nm[10];
char job_nm[10];
```

```
char sp_job_name[10];
char sp_job_number[6];
char File_Number[] = "*LAST ";
int  dlt_cnt;
char pack_dlt_count[15];
/*****************************************************************/
/* PARMS FOR QUSLSPL                                             */
/*****************************************************************/
char frmt[8];
char usr[10];
char OutQ_Nm[20];
char ls_frm_typ[10];
char Usr_dat[10];
/*****************************************************************/
/* PARMS FOR QUSRSPLA                                            */
/*****************************************************************/
char Rcv_Var[724];
int  Rcv_lgth = 724;
char Rtv_Fmt[8];
char Qal_Jb_Nam[] = "*INT                    ";
char Splf_Name[] = "*INT      ";
int  Splf_Number = -1;
/*****************************************************************/
/* PARMS FOR QUSCRTUS                                            */
/*****************************************************************/
char spc_name[20];
char ext_atr[10];
int  initial_size;
char initial_value[1];
char auth[10];
char desc[50];
/*****************************************************************/
/* PARMS FOR QMHSNDPM                                            */
/*****************************************************************/
char msg_id[7];
char msg_fl_name[20];
char msg_data[38];
int  msg_data_len;
char msg_type[10];
char msg_queue[10];
int  pgm_stk_cnt;
char msg_key[4];
/*****************************************************************/
/* END OF PARAMETERS                                             */
/*****************************************************************/
int count;
char dlt_date[7];
char spc_date[7];
char replace[10];
int  api_code;
typedef struct {
       int bytes_prov;
       int bytes_avail;
       char except_id[7];
       char reserved[1];
       char except_data[50];
       } error_code;

error_code err_code;

void strncpyn(string1,string2,strpos,cpylngth)
 char string1[],string2[];
 int strpos,cpylngth;
{
 int x = 0;
 while (x < cpylngth)
  string1[x++]=string2[strpos++];
}



main(argc,argv)
int argc;
char *argv[];
{
/*****************************************************************/
/*  Assign the parameters for the conversion routines            */
/*****************************************************************/
d_tmp_t.Type     = T_Packed;
d_tmp_t.Length   = 13;
d_tmp_t.reserved = 0;
k_tmp_t.Type     = T_Zoned;
k_tmp_t.Length   = 6;
k_tmp_t.reserved = 0;

/*****************************************************************/
/*  Read in and assign the command-line arguments to respective  */
/*  variables                                                    */
/*****************************************************************/
strncpy(spc_name,argv[1],20);
strncpy(ext_atr,argv[2],10);
initial_size = cvtcni(argv[3],d_tmp_t);
strncpy(initial_value,argv[4],1);
strncpy(auth,argv[5],10);
strncpy(desc,argv[6],50);
strncpy(frmt,argv[7],8);
strncpy(usr,argv[8],10);
strncpy(OutQ_Nm,argv[9],20);
strncpy(ls_frm_typ,argv[10],10);
strncpy(Usr_dat,argv[11],10);
strncpy(Rtv_Fmt,argv[12],8);
strncpy(dlt_date,argv[13],7);
strncpy(replace,"*NO       ",10);
/*****************************************************************/
/*  Call external program to create a user space                 */
/*****************************************************************/
err_code.bytes_prov = 16;
api_code = 1;
QUSCRTUS(spc_name,ext_atr,initial_size,initial_value,auth,desc,replace,
        &err_code);
/*****************************************************************/
/*  Call external program to get a pointer to the user space     */
/*****************************************************************/
api_code = 2;
QUSPTRUS(spc_name,&space,&err_code);
/*****************************************************************/
/*  Call external program to list spooled files into user space  */
/*****************************************************************/
api_code = 3;
QUSLSPL(spc_name,frmt,usr,OutQ_Nm,ls_frm_typ,Usr_dat,&err_code);
list_section = (char *)space;
list_section = list_section + space->list_section_offset;
entry_list = (splf0100 *) list_section;
strncpyn(dlt_spc_name,spc_name,0,10);
strncpyn(dlt_spc_lib,spc_name,10,10);
dlt_cnt = 0;
count = 1;
/*****************************************************************/
/*  Loop through the entry list and delete old spooled files     */
/*****************************************************************/
while (count <= space->number_of_entries) {
/*****************************************************************/
/*  Call external program to retrieve more spool information     */
/*****************************************************************/
 api_code = 4;
 QUSRSPLA(Rcv_Var,Rcv_lgth,Rtv_Fmt,Qal_Jb_Nam,
          entry_list->int_job_ID,entry_list->int_splf_ID,
          Splf_Name,Splf_Number,&err_code);
 Rcv_Spl_Var = (spla0100 *)Rcv_Var;
 strncpy(spc_date,Rcv_Spl_Var->date_file_open,7);
/*****************************************************************/
/*  If spooled file is too old delete it                         */
/*****************************************************************/
 if (strncmp(spc_date,dlt_date,7) <= 0 ) {
   strncpy(job_nm,Rcv_Spl_Var->job_name,10);
   strncpy(job_nmbr,Rcv_Spl_Var->job_number,6);
   strncpy(usr_nm,Rcv_Spl_Var->usr_name,10);
   strncpy(sp_job_name,Rcv_Spl_Var->splf_name,10);
   cvtnci(sp_job_number,Rcv_Spl_Var->splf_number,k_tmp_t);
   cvtnci(pack_dlt_count,++dlt_cnt,d_tmp_t);
   CLDLT(dlt_splf,dlt_spc_name,dlt_spc_lib,sp_job_name,job_nmbr,usr_nm,
         job_nm,sp_job_number,ls_frm_typ,Usr_dat,pack_dlt_count);
  } /*IF*/
 strcpy(spc_date," ");
 count++;
 entry_list++;
} /*WHILE*/
/*****************************************************************/
/*  Remove the user space                                        */
/*****************************************************************/
cvtnci(pack_dlt_count,dlt_cnt,d_tmp_t);
CLDLT(dlt_usrspc,dlt_spc_name,dlt_spc_lib,Splf_Name,job_nmbr,usr_nm,
      job_nm,File_Number,ls_frm_typ,Usr_dat,pack_dlt_count);
}  /*$USSPCEX*/
```

```
void error_check(void)
{
if (err_code.bytes_avail != 0){
     strncpy(msg_id,"CPF9898",7);
     strncpy(msg_fl_name,"QCPFMSG   *LIBL     ",20);
     strncpy(msg_data,"An error has occured calling ",29);
     switch (api_code){
        case  1  : strncat(msg_data,"QUSCRTUS.",9);
        case  2  : strncat(msg_data,"QUSPTRUS.",9);
        case  3  : strncat(msg_data,"QUSLSPL. ",9);
        case  4  : strncat(msg_data,"QUSRSPLA.",9);
        default  : strncat(msg_data,"UNKNOWN. ",9);
        }
     msg_data_len = 38;
     strncpy(msg_type,"*ESCAPE   ",10);
     strncpy(msg_queue,"*         ",10);
     pgm_stk_cnt = 1;

     QMHSNDPM(msg_id,msg_fl_name,msg_data,msg_data_len,msg_type,
             msg_queue,pgm_stk_cnt,msg_key,&err_code);
     }
}
```

To create a System C/400 PRPQ program, specify the following:

```
CRTSYSCTXT TXT(QGPL/SPOOLINFO) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/SPOOLINFO) FROMTXT(QGPL/$USSPCEX)
```

CL Delete (CLDLT) Program

The SPOOLINFO program written in RPG, COBOL, Pascal, or System C/400 PRPQ, calls a CL program named CLDLT. The CLDLT program deletes the spooled files and the user space. The following is the CL source for the CLDLT program.

```
/*                                                                */
/* ************************************************************** */
/*                                                                */
/*  PROGRAM:  CLDLT                                               */
/*                                                                */
/*  LANGUAGE:  CL                                                 */
/*                                                                */
/*  DESCRIPTION:  THIS PROGRAM WILL DELETE OLD SPOOLED FILES USING */
/*                THE DLTSPLF COMMAND, SEND MESSAGES WHEN FILES ARE */
/*                DELETED, AND DELETE A USER SPACE OBJECT.        */
/*                                                                */
/*  APIs USED:  NONE                                              */
/*                                                                */
/* ************************************************************** */
/*                                                                */
PGM (&OPTION &USNAME &USLIB &FILNAM &JOBNUM &USRNAM &JOBNAM &FILNUM +
     &FRMTYP &USRDTA &DLTCNT)
/*                                                                */
/* ************************************************************** */
/*                                                                */
/*  DECLARE SECTION                                               */
/*                                                                */
/* ************************************************************** */
/*                                                                */
 DCL &OPTION *CHAR 1
 DCL &USNAME *CHAR 10
 DCL &USLIB *CHAR 10
 DCL &FILNAM *CHAR 10
 DCL &JOBNUM *CHAR 6
 DCL &USRNAM *CHAR 10
 DCL &JOBNAM *CHAR 10
 DCL &FILNUM *CHAR 6
 DCL &FRMTYP *CHAR 10
 DCL &USRDTA *CHAR 10
 DCL &DLTCNT *DEC (15 0)
 DCL &DLTCNT1 *CHAR 15
 MONMSG CPF0000
/*                                                                */
/* ************************************************************** */
/*                                                                */
/* OPERABLE CODE                                                  */
/*                                                                */
/* ************************************************************** */
/*                                                                */
MAINLINE:

IF (&OPTION *EQ '1')   /* THIS OPTION DELETES A SPOOLED FILE */  +
   THEN(DO)
        DLTSPLF FILE(&FILNAM) JOB(&JOBNUM/&USRNAM/&JOBNAM) +
                SPLNBR(&FILNUM) SELECT(&USRNAM *ALL &FRMTYP &USRDTA)
        SNDPGMMSG  MSG('Spool file ' *CAT &JOBNUM *CAT '/' +
                       *CAT &USRNAM *CAT '/' *CAT &JOBNUM *CAT +
                       ' deleted.') TOUSR(*REQUESTER)
        ENDDO
   ELSE               /* THIS OPTION WILL DELETE THE USER SPACE  */  +
       DO
        CHGVAR &DLTCNT1 &DLTCNT
        SNDPGMMSG MSG(&DLTCNT1 *CAT ' spool files deleted.') +
                  TOUSR(*REQUESTER)
        DLTUSRSPC &USLIB/&USNAME
        SNDPGMMSG MSG('User space ' *CAT &USNAME *CAT ' in library ' +
                  *CAT &USLIB *CAT ' deleted.') TOUSR(*REQUESTER)
       ENDDO
ENDPGM
```

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CLDLT) SRCFILE(QGPL/QCLSRC)
```

Changing an Active Job

This command interface to the Change Active Jobs (CHGACTJOB) program can reduce the run priority of active jobs with the same name. You can also reduce the run priority of jobs using a specified user name. You may:

- Specify a job name or the *ALL value.
- Specify the user name as the *ALL value.
- Use the default run priority of 99.

The CHGACTJOB command ensures that one of the following is true:

- Not all jobs were specified.
- The *ALL value was not specified for the JOB parameter.
- The *ALL value was not specified for the USER parameter.

This example uses the following APIs:

- Create User Space (QUSCRTUS)
- List Job (QUSLJOB)
- Retrieve User Space (QUSRTVUS)
- Retrieve Job Information (QUSRJOBI)

The following is the message description needed for the Change Active Jobs (CHGACTJOB) command:

```
ADDMSGD MSGID(USR3C01) MSGF(QCPFMSG) +
MSG('JOB(*ALL) is not valid with USER(*ALL)') SEV(30)
```

The following is the command definition for the CHGACTJOB command:

```
     CMD        PROMPT('Change Active Jobs')
                /* CPP CHGACTJOB */
     PARM       KWD(JOB) TYPE(*NAME) LEN(10) +
                  SPCVAL((*ALL)) MIN(1) +
                  PROMPT('Job name:')
     PARM       KWD(USER) TYPE(*NAME) LEN(10) DFT(*ALL) +
                  SPCVAL((*ALL) (*CURRENT)) PROMPT('User +
                  name:')
     PARM       KWD(RUNPTY) TYPE(*DEC) LEN(5 0) DFT(99) +
                  RANGE(00 99) PROMPT('Run priority:')
     DEP        CTL(&USER *EQ *ALL) PARM((&JOB *NE *ALL)) +
                  NBRTRUE(*EQ 1) MSGID(USR3C01)
```

To create the command, specify the following:

```
CRTCMD CMD(QGPL/CHGACTJOB) PGM(QGPL/CHGACTJOB) +
  SRCFILE(QGPL/CMDSRC)
```

The following is the command-processing program that is written in CL to list the active jobs and reduce the run priority if necessary:

```
/* ************************************************************ */
/* PROGRAM:  CHGACTJOB                                          */
/*                                                              */
/* LANGUAGE:  CL                                                */
/*                                                              */
/* DESCRIPTION:  THIS PROGRAM WILL REDUCE THE RUN PRIORITY OF ACTIVE */
/*               JOBS WITH THE SAME NAME.                       */
/*                                                              */
/* APIs USED:  QUSCRTUS, QUSLJOB, QUSRTVUS, QUSRJOBI            */
/*                                                              */
/* ************************************************************ */
            PGM         PARM(&JOB &USER &RUNPTY)

/*                                                              */
/* Input parameters                                             */
/*                                                              */

            DCL         VAR(&JOB) TYPE(*CHAR) LEN(10) +
                          /* Input job name */
            DCL         VAR(&USER) TYPE(*CHAR) LEN(10) +
                          /* Input user name */
            DCL         VAR(&RUNPTY) TYPE(*DEC) LEN(5 0) +
                          /* Input run priority */

/*                                                              */
/* Local variables                                              */
/*                                                              */

            DCL         VAR(&RJOB) TYPE(*CHAR) LEN(10) +
                          /* Retrieve job name */
            DCL         VAR(&RUSER) TYPE(*CHAR) LEN(10) +
                          /* Retrieve user name */
            DCL         VAR(&RNBR) TYPE(*CHAR) LEN(6) +
                          /* Retrieve job number */
            DCL         VAR(&RUNPTYC) TYPE(*CHAR) LEN(5) +
                          /* Input run priority in character form */
            DCL         VAR(&RUNPTY8) TYPE(*DEC) LEN(8 0) +
                          /* Retrieve run priority after convert from +
                             binary 4 */
            DCL         VAR(&RUNPTY5) TYPE(*DEC) LEN(5 0) +
                          /* Retrieve run priority in decimal 5,0 +
                             form */
            DCL         VAR(&RUNPTY5C) TYPE(*CHAR) LEN(5) +
                          /* Retrieve run priority in character form */
            DCL         VAR(&RUNPTY4) TYPE(*CHAR) LEN(4) +
                          /* Retrieve run priority in binary 4 form */
            DCL         VAR(&NUMBER) TYPE(*CHAR) LEN(6) +
                          /* Current job number */
            DCL         VAR(&USRSPC) TYPE(*CHAR) LEN(20) +
                          VALUE('CHGA      QTEMP     ') +
                          /* User space name for APIs */
            DCL         VAR(&EUSRSPC) TYPE(*CHAR) LEN(10) +
                          /* User space name for commands */
            DCL         VAR(&JOBNAME) TYPE(*CHAR) LEN(26) +
                          VALUE('                    *ALL  ') +
                          /* Full job name for list job  */
            DCL         VAR(&BIN4) TYPE(*CHAR) LEN(4) +
                          /* Number of jobs for list job and +
                             User space offset in binary 4 form */
            DCL         VAR(&LOOP) TYPE(*DEC) LEN(8 0) +
                          /* Number of jobs from list job */
            DCL         VAR(&DEC8) TYPE(*DEC) LEN(8 0) +
                          /* User space offset in decimal 8,0 form */
            DCL         VAR(&ELEN) TYPE(*DEC) LEN(8 0) +
                          /* List job entry length in decimal 8,0 +
                             form */
            DCL         VAR(&ELENB) TYPE(*CHAR) LEN(4) +
                          /* List job entry length in binary 4 +
                             form */
            DCL         VAR(&LJOBE) TYPE(*CHAR) LEN(52) +
                          /* Retrieve area for list job entry */
            DCL         VAR(&INTJOB) TYPE(*CHAR) LEN(16) +
                          /* Retrieve area for internal job id */
            DCL         VAR(&JOBI) TYPE(*CHAR) LEN(104) +
                          /* Retrieve area for job information */
            DCL         VAR(&JOBTYPE) TYPE(*CHAR) LEN(1) +
                          /* Job type */

/*                                                              */
/* Start of executable code                                     */
/*                                                              */

/*                                                              */
/* Retrieve job number to use for local user space name         */
/*                                                              */

            RTVJOBA     NBR(&NUMBER)
            CHGVAR      VAR(%SST(&USRSPC 5 6)) VALUE(&NUMBER)
            CHGVAR      VAR(&EUSRSPC) VALUE(%SST(&USRSPC 1 10))

/*                                                              */
/* Delete user space if it already exists                       */
/*                                                              */

            DLTUSRSPC  USRSPC(QTEMP/&EUSRSPC)
            MONMSG CPF0000

/*                                                              */
/* Create user space                                            */
/*                                                              */

            CALL QUSCRTUS (&USRSPC 'CHGACTJOB ' X'00000100' ' ' +
                          '*ALL      ' +
                          'CHGACTJOB TEMPORARY USER SPACE-
                ')

/*                                                              */
/* Set up job name for list jobs                                */
/*                                                              */

            CHGVAR      VAR(%SST(&JOBNAME 1 10)) VALUE(&JOB)
            CHGVAR      VAR(%SST(&JOBNAME 11 10)) VALUE(&USER)

/*                                                              */
/* List active jobs with job name specified                     */
/*                                                              */

            CALL QUSLJOB  (&USRSPC 'JOBL0100' &JOBNAME +
                          '*ACTIVE   ')

/*                                                              */
/* Retrieve number of entries returned. Convert to decimal and  */
/* if zero go to NOJOBS label to send out 'No jobs' message.    */
/*                                                              */

            CALL QUSRTVUS (&USRSPC X'00000085' X'00000004' +
                          &BIN4)
            CHGVAR      &LOOP      %BINARY(&BIN4)
            IF          COND(&LOOP = 0) THEN(GOTO CMDLBL(NOJOBS))

/*                                                              */
/* Retrieve list entry length, convert to decimal.              */
/* Retrieve list entry offset, convert to decimal, and add one  */
/* to set the position.                                         */
/*                                                              */

            CALL QUSRTVUS (&USRSPC X'00000089' X'00000004' +
                          &ELENB)
            CHGVAR      &ELEN      %BINARY(&BIN4)
            CALL QUSRTVUS (&USRSPC X'0000007D' X'00000004' +
                          &BIN4)
            CHGVAR      &DEC8      %BINARY(&BIN4)
            CHGVAR      VAR(&DEC8) VALUE(&DEC8 + 1)

/*                                                              */
/* Loop for the number of jobs until no more jobs then go to    */
/* ALLDONE label                                                */
/*                                                              */

STARTLOOP:  IF (&LOOP = 0) THEN(GOTO ALLDONE)

/*                                                              */
/* Convert decimal position to binary 4 and retrieve list job entry */
/*                                                              */

            CHGVAR      %BINARY(&BIN4)   &DEC8
            CALL QUSRTVUS (&USRSPC &BIN4 &ELENB +
```

```
                        &LJOBE)

/*                                                                  */
/* Copy internal job identifier and retrieve job information for    */
/* basic performance information.                                   */
/*                                                                  */

            CHGVAR      VAR(&INTJOB) VALUE(%SST(&LJOBE 27 16))
            CALL QUSRJOBI (&JOBI X'00000068' 'JOBI0100' +
                           '*INT                    ' +
                           &INTJOB)

/*                                                                  */
/* Copy job type and if subsystem monitor, spool reader, system job, */
/* spool writer, or SCPF system job then loop to next job           */
/*                                                                  */

            CHGVAR      VAR(&JOBTYPE) VALUE(%SST(&JOBI 61 1))
            IF          COND((&JOBTYPE = 'M') *OR (&JOBTYPE = 'R')  +
                         *OR (&JOBTYPE = 'S') *OR (&JOBTYPE = 'W')  +
                         *OR (&JOBTYPE = 'X')) +
                          THEN(GOTO CMDLBL(ENDLOOP))

/*                                                                  */
/* Copy run priority, convert to decimal, convert to decimal 5,0,   */
/* and if request run priority is less than or equal to the current */
/* run priority then loop to next job.                              */
/*                                                                  */

            CHGVAR      VAR(&RUNPTY4) VALUE(%SST(&JOBI 65 4))
            CHGVAR      &RUNPTY8      %BINARY(&RUNPTY4)
            CHGVAR      VAR(&RUNPTY5) VALUE(&RUNPTY8)
            IF          COND(&RUNPTY5 *GE &RUNPTY) THEN(GOTO +
                          CMDLBL(ENDLOOP))

/*                                                                  */
/* Retrieve job name, convert to run priority to character, change  */
/* the job run priority and seen message stating the run priority   */
/* was changed.                                                     */
/*                                                                  */

            CHGVAR      VAR(&RJOB) VALUE(%SST(&JOBI 9 10))
            CHGVAR      VAR(&RUSER) VALUE(%SST(&JOBI 19 10))
            CHGVAR      VAR(&RNBR) VALUE(%SST(&JOBI 29 6))
            CHGVAR      VAR(&RUNPTYC) VALUE(&RUNPTY)
            CHGVAR      VAR(&RUNPTY5C) VALUE(&RUNPTY5)
            CHGJOB      JOB(&RNBR/&RUSER/&RJOB) RUNPTY(&RUNPTYC)
            MONMSG      MSGID(CPF1343) EXEC(GOTO CMDLBL(ENDLOOP))
            SNDPGMMSG   MSG('Job' *BCAT &RNBR *TCAT '/' *TCAT +
                          &RUSER *TCAT '/' *TCAT &RJOB *BCAT 'run +
                          priority was change from' *BCAT &RUNPTY5C +
                          *BCAT 'to' *BCAT &RUNPTYC *TCAT '.')

/*                                                                  */
/* At end of loop set new decimal position to next entry and        */
/* decrement loop counter by one.                                   */
/*                                                                  */

ENDLOOP:    CHGVAR      VAR(&DEC8) VALUE(&DEC8 + &ELEN)
            CHGVAR      VAR(&LOOP) VALUE(&LOOP - 1)
            GOTO        CMDLBL(STARTLOOP)

/*                                                                  */
/* Send message that no jobs were found.                            */
/*                                                                  */

NOJOBS:     SNDPGMMSG   MSG('No jobs found.')

/*                                                                  */
/* All done. Now delete temporary user space that we created.       */
/*                                                                  */

ALLDONE:    DLTUSRSPC   USRSPC(QTEMP/&EUSRSPC)
            MONMSG CPF0000
            ENDPGM
```

The program can be changed to change the run priority by removing the IF statement to compare the current and requested run priority.

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CHGACTJOB) SRCFILE(QGPL/QCLSRC)
```

You can change the command to:

- Specify a different printer device.
- Specify a different output queue.
- Specify different job attributes that the Change Job (CHGJOB) command can change.
- List only jobs on an output queue and remove the spooled files.
- Provide a menu to select jobs to be changed.

Changing a Job Schedule Entry

This command interface to the Change Job Schedule Entry User (CHGSCDEUSR) program can change the USER parameter in the job schedule entry. You may:

- Specify a job schedule entry name
- Specify a generic job schedule entry name
- Specify the *ALL value

This example uses the following APIs:

- Create User Space (QUSCRTUS)
- List Job Schedule Entries (QWCLSCDE)
- Retrieve User Space (QUSRTVUS)

The following is the command definition for the CHGSCDEUSR command:

```
CMD         PROMPT('Change Job Schedule Entry User')
            /* CPP CHGSCDEUSR */
PARM        KWD(JOB) TYPE(*GENERIC) LEN(10) +
              SPCVAL((*ALL)) +
              MIN(1) PROMPT('Job name:')
PARM        KWD(OLDUSER) TYPE(*NAME) LEN(10) +
              MIN(1) PROMPT('Old user name:')
PARM        KWD(NEWUSER) TYPE(*NAME) LEN(10) +
              MIN(1) PROMPT('New user name:')
```

To create the command, specify the following:

```
CRTCMD CMD(QGPL/CHGSCDEUSR) PGM(QGPL/CHGSCDEUSR) +
  SRCFILE(QGPL/QCMDSRC)
```

The following is the command-processing program that is written in CL to list the job schedule entries and change the user if necessary:

```
/* ************************************************************ */
/* PROGRAM:  CHGSCDEUSR                                         */
/*                                                              */
/* LANGUAGE:  CL                                                */
/*                                                              */
/* DESCRIPTION:  THIS PROGRAM WILL CHANGE THE USER FOR A LIST OF */
/*               JOB SCHEDULE ENTRIES.                          */
/*                                                              */
/* APIs USED:  QUSCRTUS, QWCLSCDE, QUSRTVUS                     */
/*                                                              */
/* ************************************************************ */
            PGM         PARM(&JOBNAME &OLDUSER &NEWUSER)

/*                                                              */
/* Input parameters are as follows:                             */
/*                                                              */

            DCL         VAR(&JOBNAME) TYPE(*CHAR) LEN(10) /* Input +
                          job name */
            DCL         VAR(&OLDUSER) TYPE(*CHAR) LEN(10) /* Input +
                          old user name */
            DCL         VAR(&NEWUSER) TYPE(*CHAR) LEN(10) /* Input +
                          new user name */

/*                                                              */
/* Local variables are as follows:                              */
/*                                                              */
```

```
            DCL         VAR(&USRSPC) TYPE(*CHAR) LEN(20) +
                          VALUE('CHGSCDEUSRQTEMP     ') /* User +
                          space name for APIs */
            DCL         VAR(&CNTHDL) TYPE(*CHAR) LEN(16) +
                          VALUE('                ') /* Continuation +
                          handle */
            DCL         VAR(&NUMENTB) TYPE(*CHAR) LEN(4) /* Number +
                          of entries from list job schedule entries +
                          in binary form */
            DCL         VAR(&NUMENT) TYPE(*DEC) LEN(8 0) /* Number +
                          of entries from list job schedule entries +
                          in decimal form */
            DCL         VAR(&HDROFFB) TYPE(*CHAR) LEN(4) /* Offset +
                          to the header portion of the user space in +
                          binary form */
            DCL         VAR(&HDRLENB) TYPE(*CHAR) LEN(4) /* Length +
                          to the header portion of the user space in +
                          binary form */
            DCL         VAR(&GENHDR) TYPE(*CHAR) LEN(140) /* Generic +
                          header information from the user space */
            DCL         VAR(&HDRINFO) TYPE(*CHAR) LEN(26) /* Header +
                          information from the user space */
            DCL         VAR(&LSTSTS) TYPE(*CHAR) LEN(1) /* Status +
                          of the list in the user space */
            DCL         VAR(&OFFSETB) TYPE(*CHAR) LEN(4) /* Offset +
                          to the list portion of the user space in +
                          binary form */
            DCL         VAR(&STRPOSB) TYPE(*CHAR) LEN(4) /* Starting +
                          position in the user space  in binary form */
            DCL         VAR(&ELENB) TYPE(*CHAR) LEN(4) /* List job +
                          entry length in binary 4 form */
            DCL         VAR(&LENTRY) TYPE(*CHAR) LEN(1156) /* +
                          Retrieve area for list job schedule entry */
            DCL         VAR(&INFOSTS) TYPE(*CHAR) LEN(1) /* Retrieve +
                          area for information status */
            DCL         VAR(&JOBNAM) TYPE(*CHAR) LEN(10) /* Retrieve +
                          area for job name */
            DCL         VAR(&ENTRY#) TYPE(*CHAR) LEN(6) /* Retrieve +
                          area for entry number  */
            DCL         VAR(&USERNM) TYPE(*CHAR) LEN(10) /* Retrieve +
                          area for user name */

/*                                                                  */
/* Start of code                                                    */
/*                                                                  */

/*                                                                  */
/* You may want to monitor for additional messages here.            */
/*                                                                  */

/*                                                                  */
/* This creates the user space.  The user space will be 256 bytes   */
/* and will be initialized to hexidecimal zeros.                    */
/*                                                                  */

            CALL        PGM(QUSCRTUS) PARM(&USRSPC 'CHGSCDEUSR' +
                          X'00000100' ' ' '*ALL      ' 'CHGSCDEUSR +
                          TEMPORARY USER SPACE                  ')
            MONMSG      MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))

/*                                                                  */
/* This lists job schedule entries of the name specified.           */
/*                                                                  */

PARTLIST:   CALL        PGM(QWCLSCDE) PARM(&USRSPC 'SCDL0200' +
                          &JOBNAME &CNTHDL 0)

/*                                                                  */
/* Retrieve the generic header from the user space.                 */
/*                                                                  */

            CALL        PGM(QUSRTVUS) PARM(&USRSPC X'00000001' +
                          X'0000008C' &GENHDR)
            MONMSG      MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))
/*                                                                  */
/* Get the information status for the list from the generic header. */
/* If it is incomplete, go to BADLIST label and send out 'Bad list' */
/* message.                                                         */
/*                                                                  */

            CHGVAR      VAR(&LSTSTS) VALUE(%SST(&GENHDR 104 1))
            IF          COND(&LSTSTS = 'I') THEN(GOTO CMDLBL(BADLIST))

/*                                                                  */
/* Get the number of entries returned. Convert to decimal and       */
/* if zero go to NOENTS label to send out 'No entries' message.     */
/*                                                                  */

            CHGVAR      VAR(&NUMENTB) VALUE(%SST(&GENHDR 133 4))
            CHGVAR      VAR(&NUMENT) VALUE(%BIN(&NUMENTB))
            IF          COND(&NUMENT = 0) THEN(GOTO CMDLBL(NOENTS))

/*                                                                  */
/* Get the list entry length and the list entry offset.             */
/* These values are used to set up the starting position.           */
/*                                                                  */

            CHGVAR      VAR(&ELENB) VALUE(%SST(&GENHDR 137 4))
            CHGVAR      VAR(&OFFSETB) VALUE(%SST(&GENHDR 125 4))
            CHGVAR      VAR(%BIN(&STRPOSB)) VALUE(%BIN(&OFFSETB) + 1)

/*                                                                  */
/* This loops for the number of entries until no more entries are   */
/* found and goes to the ALLDONE label.                             */
/*                                                                  */

STARTLOOP:  IF          COND(&NUMENT = 0) THEN(GOTO CMDLBL(PARTCHK))

/*                                                                  */
/* This retrieves the list entry.                                   */
/*                                                                  */
            CALL        PGM(QUSRTVUS) PARM(&USRSPC &STRPOSB &ELENB +
                          &LENTRY)
            MONMSG      MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))
/*                                                                  */
/* This copies the information status, job name, entry number, and  */
/* user name.                                                       */
/*                                                                  */

            CHGVAR      VAR(&INFOSTS) VALUE(%SST(&LENTRY 1 1))
            CHGVAR      VAR(&JOBNAM)  VALUE(%SST(&LENTRY 2 10))
            CHGVAR      VAR(&ENTRY#)  VALUE(%SST(&LENTRY 12 10))
            CHGVAR      VAR(&USERNM)  VALUE(%SST(&LENTRY 547 10))

/*                                                                  */
/* This checks to make sure the list entry contains the user name.  */
/* If it does, the user name is compared to the old user name       */
/* passed in.  If either of these checks fails, this entry will     */
/* be skipped.                                                      */
/*                                                                  */

            IF          COND(&INFOSTS ¬= ' ') THEN(GOTO +
                          CMDLBL(ENDLOOP))

            IF          COND(&USERNM ¬= &OLDUSER) THEN(GOTO +
                          CMDLBL(ENDLOOP))

/*                                                                  */
/* This code will issue the CHGJOBSCDE command for the entry.       */
/*                                                                  */

            CHGJOBSCDE JOB(&JOBNAM) ENTRYNBR(&ENTRY#) USER(&NEWUSER)
            MONMSG      MSGID(CPF1620) EXEC(GOTO CMDLBL(NOCHG))
            SNDPGMMSG   MSG('Entry' *BCAT &JOBNAM *BCAT &ENTRY# +
                          *BCAT 'was changed.')
            GOTO        CMDLBL(ENDLOOP)
NOCHG:      SNDPGMMSG   MSG('Entry' *BCAT &JOBNAM *BCAT &ENTRY# +
                          *BCAT 'was NOT changed.')

/*                                                                  */
/* At end of loop, set new decimal position to the next entry and   */
/* decrement the loop counter by one.                               */
/*                                                                  */

ENDLOOP:    CHGVAR      VAR(%BIN(&STRPOSB)) VALUE(%BIN(&STRPOSB) +
                          + %BIN(&ELENB))
            CHGVAR      VAR(&NUMENT) VALUE(&NUMENT - 1)
            GOTO        CMDLBL(STARTLOOP)

/*                                                                  */
/* This sends a message that no entries were found.                 */
/*                                                                  */

NOENTS:     SNDPGMMSG   MSG('No entries found.')
            GOTO        CMDLBL(ALLDONE)
```

```
/*                                                                  */
/* This sends a message that the list was incomplete.               */
/*                                                                  */

BADLIST:    SNDPGMMSG  MSG('Incomplete list in the user space. +
                         See joblog for details.')
            GOTO       CMDLBL(ALLDONE)

/*                                                                  */
/* This sends a message that an unexpected error occurred.          */
/*                                                                  */

ERROR:      SNDPGMMSG  MSG('Unexpected error. +
                         See joblog for details.')
            GOTO       CMDLBL(ALLDONE)

/*                                                                  */
/* This will check for a partial list in the user space and         */
/* finish processing the rest of the list.                          */
/*                                                                  */

PARTCHK:    IF         COND(&LSTSTS = 'C') THEN(GOTO CMDLBL(ALLDONE))
/*                                                                  */
/* Retrieve the header information from the user space.             */
/* Use this information to get the rest of the list.                */
/*                                                                  */

            CHGVAR     VAR(&HDROFFB) VALUE(%SST(&GENHDR 121 4))
            CHGVAR     VAR(&HDRLENB) VALUE(%SST(&GENHDR 117 4))
            CALL       PGM(QUSRTVUS) PARM(&USRSPC &HDROFFB +
                         &HDRLENB &HDRINFO)
            MONMSG     MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))
            CHGVAR     VAR(&CNTHDL) VALUE(%SST(&HDRINFO 11 16))
            GOTO       CMDLBL(PARTLIST)


/*                                                                  */
/* All done.  Now the temporary user space is deleted.              */
/*                                                                  */

ALLDONE:    DLTUSRSPC  USRSPC(QTEMP/%SST(&USRSPC 1 10))
            MONMSG     MSGID(CPF0000)
            ENDPGM
```

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CHGSCDEUSR) SRCFILE(QGPL/QCLSRC)
```

You can change the command to:

- Specify different parameters that the Change Job Schedule Entry (CHGJOBSCDE) command can change.
- Provide a menu to select job schedule entries to be changed.

Creating Your Own Telephone Directory

To create a telephone directory, you must use the following $USIDXCRT System C/400 PRPQ program to create a user index, and you must use the $USIDXEX System C/400 PRPQ program to insert the entries into a telephone directory.

To set up the program to create a user index, specify the following:

```
/*********************************************************************/
/*  PROGRAM:  $USIDXCRT                                              */
/*                                                                   */
/*  LANGUAGE:  System C/400 PRPQ                                     */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM CREATES A USER INDEX NAMED TESTIDX    */
/*    IN THE LIBRARY QGPL.                                           */
/*                                                                   */
/*  APIs USED:  QUSCRTUI                                             */
/*                                                                   */
/*********************************************************************/
#include "OPENAPI.H"   /* Linkage info, structures     */
main()
{
/*********************************************************************/
/*  This sets up the parameters to call the QUSCRTUI API.            */
/*********************************************************************/
char idx_name[]= "TESTIDX   QGPL      ";
char ext_atr[]= "TESTER    ";
char entry_lgth_att[] = "F";
int entry_lngth = 50;
char key_insert[] = "1";
int key_lngth = 15;
char imm_update[] = "0";
char optim[] = "0";
char auth[] = "*CHANGE   ";
char desc[] = "Description .....    ";
/*********************************************************************/
/*  This calls the QUSCRTUI API to create the user index.            */
/*********************************************************************/
  QUSCRTUI(idx_name,ext_atr,entry_lgth_att,entry_lngth,key_insert,
           key_lngth,imm_update,optim,auth,desc);
}
```

To compile the program that creates the user index, specify the following:

```
CRTSYSCTXT TXT(QGPL/$USIDXCRT) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/$USIDXCRT) FROMTXT(QGPL/$USIDXCR)
```

To insert entries into the user index, use the following System C/400 PRPQ program:

```
/*********************************************************************/
/*  PROGRAM:  $USIDXEX                                               */
/*                                                                   */
/*  LANGUAGE:  System C/400 PRPQ                                     */
/*                                                                   */
/*  DESCRIPTION:  This program uses a user index to keep track of    */
/*    names and telephone numbers.  Two operations are shown in      */
/*    this example.  The first operation inserts an entry into       */
/*    the user index, and the second operation finds the index       */
/*    entry.                                                         */
/*    The user index is keyed on the last name, so you can enter as  */
/*    many of the names as you know, and the program lists all the   */
/*    entries matching your string in alphabetic order.              */
/*                                                                   */
/*  APIs USED:  None                                                 */
/*                                                                   */
/*********************************************************************/
#define C400
#include "C400.h" /* printf,gets,clibinit,clibterm,_Usridx,_AUTH_ALL*/
#include <string.h>   /*strlen, strcpy, strncmp          */
#include <miindex.h>  /*_IIX_Opt_List_T,insinxen,fndinxen*/
#include <miptrnam.h> /*_SYSPTR, rslvsp                  */
#include <stdlib.h>   /*malloc   */
#include <ctype.h>    /*toupper  */

_SYSPTR index;
_IIX_Opt_List_T  ins_option_list;
_IIX_Opt_List_T  *fnd_option_list;
char Name_And_Num[50];
char In_Name[50];
char Out_Num[5000];
char response[1];
char name[35];
char number[15];
int  Ent_Found,count,start,length_of_entry;


/*********************************************************************/
/* This copies the cpylngth elements of string2 into the            */
/* new string, string1, starting from position strpos.              */
/*********************************************************************/
void strncpyn(string1,string2,strpos,cpylngth)
 char string1[],string2[];
 int strpos,cpylngth;
{
 int x = 0;
 while (x < cpylngth)
  string1[x++]=string2[strpos++];
} /*strncpyn*/
```

```
/*********************************************************************/
/* This converts any string into uppercase, where it is applicable. */
/*********************************************************************/
void convert_case(string1)
 char string1[];
{
 int x = 0;
 while (x < (strlen(string1))) {
  string1[x] = toupper(string1[x]);
  x++;
 } /*while*/
} /*convert_case*/


main()
{
fnd_option_list =
malloc(sizeof(_IIX_Opt_List_T)+99*sizeof(_IIX_Entry_T));
/*********************************************************************/
/*  This resolves to the index created in $USIDXCRT.                 */
/*********************************************************************/
  index = rslvsp(_Usridx,"TESTIDX","QGPL",_AUTH_ALL);
/*********************************************************************/
/*  This sets up the insert option list.                             */
/*********************************************************************/
  ins_option_list.Rule = _Insert_Repl;
  ins_option_list.Arg_Length = 50;
  ins_option_list.Occ_Count = 1;
  ins_option_list.Entry[0].Entry_Length = 50;
  ins_option_list.Entry[0].Entry_Offset = 0;
/*********************************************************************/
/*  This sets up the find option list.                               */
/*********************************************************************/
  fnd_option_list->Rule = _Equals;
  fnd_option_list->Occ_Count = 100;
/*********************************************************************/
/*  This establishes a linkage to a C/400 program which does I/O     */
/*  operations.                                                      */
/*********************************************************************/
clibinit;
/*********************************************************************/
/*  This loops until the choice 'Q' is entered at the menu.          */
/*********************************************************************/
 while (1==1) {
  printf("\n\n**********************\n");
  printf("*   TELEPHONE INDEX   *\n");
  printf("**********************\n");
  printf("* 'A'  Add name & num *\n");
  printf("* 'L'  List a number  *\n");
  printf("* 'Q'  Quit index     *\n");
  printf("**********************\n");
  gets(response);
  if ((strncmp(response,"A",1)==0)||(strncmp(response,"a",1)==0))
   { printf("\nEnter name to add. ex(Last, First)\n");
     gets(name);
     convert_case(name);
     printf("\nEnter number to add. ex(999-9999)\n");
     gets(number);
     strcpy(name,strcat(name,"     "));
     strcpy(Name_And_Num,strcat(name,number));
     printf("\nName and number to add is => %s\n",Name_And_Num);
     insinxen(index,Name_And_Num,&ins_option_list);
   } /* if 'a'*/
  if ((strncmp(response,"L",1)==0)||(strncmp(response,"l",1)==0))
   {
     printf("\nEnter name to find. ex(Last, First)\n");
     gets(In_Name);
     convert_case(In_Name);
     fnd_option_list->Arg_Length = strlen(In_Name);
     fndinxen(Out_Num,index,fnd_option_list,In_Name);
     length_of_entry = fnd_option_list->Entry[0].Entry_Length;
     Ent_Found = fnd_option_list->Ret_Count;
     if (Ent_Found == 0)
       printf("\nName not found in index => %s\n",In_Name);
     else {
       if (Ent_Found > 1)  {
        printf("\n%d occurences found,\n",Ent_Found);
        count = 0;
        start = 0;
        while (count++ < Ent_Found) {
         printf("Name and number is => %s\n",Out_Num);
         start = start + length_of_entry;
         strncpyn(Out_Num,Out_Num,start,length_of_entry);
       } /* while */
      }else
       printf("\nName and number is => %s\n",Out_Num);
    }  /*else*/
  }  /*if 'l'*/
 if ((strncmp(response,"Q",1)==0)||(strncmp(response,"q",1)==0))
  { break; }
 }  /*while*/
clibterm;
} /*$USIDXEX*/
```

To create the System C/400 PRPQ program to insert entries into the user index, specify the following:

```
CRTSYSCTXT TXT(QGPL/$USIDXEX) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/$USIDXEX) FROMTXT(QGPL/$USIDXEX)
```

Creating and Manipulating a User Index

The following example shows how to create and manipulate a user index with a call from an MI program. For another example using the QUSCRTUI API, see "Creating Your Own Telephone Directory" on page A-13.

```
/*********************************************************************/
/*                                                                   */
/* PROGRAM:  GLOBALV                                                 */
/*                                                                   */
/* LANGUAGE:  MI/IRP                                                 */
/*                                                                   */
/* DESCRIPTION:  MAINTAINS AN INDEPENDENT INDEX.  EACH INDEX ENTRY   */
/*               CONTAINS 100 BYTES OF USER DATA.  THE ENTRIES ARE   */
/*               KEYED TWO 10 BYTE VALUES:  THE USER PROFILE AND A   */
/*               VALUE IDENTIFIER.                                   */
/*                                                                   */
/* APIs USED:  QUSCRTUI                                              */
/*                                                                   */
/* PARAMETERS:                                                       */
/*                                                                   */
/*   PARM  TYPE       DESCRIPTION                                    */
/*                                                                   */
/*    1    CHAR(1)    FUNCTION:                                      */
/*                                                                   */
/*                    'U':  UPDATE GLOBALV INFORMATION               */
/*                    'R':  RETRIEVE GLOBALV INFORMATION             */
/*                                                                   */
/*    2    CHAR(10)   USER PROFILE                                   */
/*                                                                   */
/*                    THE NAME OF THE USER PROFILE FOR WHICH         */
/*                    INFORMATION IS TO BE SAVED OR RETRIEVED.       */
/*                                                                   */
/*    3    CHAR(10)   VALUE ID                                       */
/*                                                                   */
/*                    THE NAME OF THE GLOBALV VARIABLE ID FOR WHICH  */
/*                    INFORMATION IS TO BE SAVED OR RETRIEVED.       */
/*                                                                   */
/*    4    CHAR(100)  VALUE                                          */
/*                                                                   */
/*                    IF FUNCTION IS 'U', THIS VALUE SHOULD CONTAIN  */
/*                    THE NEW VALUE TO BE ASSOCIATED WITH THE        */
/*                    USER ID AND VALUE ID.                          */
/*                                                                   */
/*                    IF FUNCTION IS 'R', THIS VARIABLE WILL BE      */
/*                    SET TO THE VALUE ASSOCIATED WITH THE USER ID   */
/*                    AND VALUE ID.  IF NO VALUE EXISTS, *NONE       */
/*                    IS SPECIFIED.                                  */
/*                                                                   */
/*********************************************************************/

ENTRY * (GLOBALV_PARM) EXT;

/*********************************************************************/
/* PARAMETER VALUE POINTERS FOR GLOBALV.                             */
/*********************************************************************/

DCL SPCPTR GV_REQUEST@ PARM;
DCL SPCPTR GV_USERID@ PARM;
DCL SPCPTR GV_VALUEID@ PARM;
DCL SPCPTR GV_VALUE@ PARM;
```

```
/*********************************************************************/
/* PARAMETER VALUES FOR GLOBALV.                                     */
/*********************************************************************/

DCL DD GV_REQUEST CHAR(1) BAS(GV_REQUEST@);
DCL DD GV_USERID CHAR(10) BAS(GV_USERID@);
DCL DD GV_VALUEID CHAR(10) BAS(GV_VALUEID@);
DCL DD GV_VALUE CHAR(100) BAS(GV_VALUE@);

/*********************************************************************/
/* PARAMETER LIST FOR GLOBALV.                                       */
/*********************************************************************/

DCL OL GLOBALV_PARM (GV_REQUEST@
                    ,GV_USERID@
                    ,GV_VALUEID@
                    ,GV_VALUE@
                    ) PARM EXT;

/*********************************************************************/
/* ARGUMENT VALUES FOR CREATE USER INDEX (QUSCRTUI) API.             */
/*********************************************************************/

DCL DD UI_NAME CHAR(20) INIT("GLOBALV   QGPL      ");
DCL DD UI_ATTR CHAR(10) INIT("          ");
DCL DD UI_EATR CHAR(1) INIT("F");
DCL DD UI_ELEN BIN(4) INIT(120);
DCL DD UI_KATR CHAR(1) INIT("1");
DCL DD UI_KLEN BIN(4) INIT(20);
DCL DD UI_IUPD CHAR(1) INIT("0");
DCL DD UI_OPT  CHAR(1) INIT("0");
DCL DD UI_AUT  CHAR(10) INIT("*CHANGE   ");
DCL DD UI_TEXT CHAR(50)
       INIT("GLOBALV INDEX                                ");

/*********************************************************************/
/* POINTERS TO ARGUMENT VALUES FOR QUSCRTUI API.                     */
/*********************************************************************/

DCL SPCPTR UI_NAME@ INIT(UI_NAME);
DCL SPCPTR UI_ATTR@ INIT(UI_ATTR);
DCL SPCPTR UI_EATR@ INIT(UI_EATR);
DCL SPCPTR UI_ELEN@ INIT(UI_ELEN);
DCL SPCPTR UI_KATR@ INIT(UI_KATR);
DCL SPCPTR UI_KLEN@ INIT(UI_KLEN);
DCL SPCPTR UI_IUPD@ INIT(UI_IUPD);
DCL SPCPTR UI_OPT@  INIT(UI_OPT);
DCL SPCPTR UI_AUT@  INIT(UI_AUT);
DCL SPCPTR UI_TEXT@ INIT(UI_TEXT);

/*********************************************************************/
/* ARGUMENT LIST FOR QUSCRTUI API.                                   */
/*********************************************************************/

DCL OL QUSCRTUI_ARG (UI_NAME@
                    ,UI_ATTR@
                    ,UI_EATR@
                    ,UI_ELEN@
                    ,UI_KATR@
                    ,UI_KLEN@
                    ,UI_IUPD@
                    ,UI_OPT@
                    ,UI_AUT@
                    ,UI_TEXT@
                    ) ARG;

/*********************************************************************/
/* SYTSEM POINTER TO QUSCRTUI API *PGM OBJECT.                       */
/*********************************************************************/

DCL SYSPTR QUSCRTUI INIT("QUSCRTUI",TYPE(PGM));

/*********************************************************************/
/* SYSTEM POINTER TO GLOBALV *USRIDX OBJECT.                         */
/*********************************************************************/

DCL SYSPTR INX@;

DCL DD INX_OBJECTID CHAR(34);
DCL DD INX_OBJECTID_TYPE CHAR(2) DEF(INX_OBJECTID) POS(1)
       INIT(X'0E0A');
DCL DD INX_OBJECTID_NAME CHAR(30) DEF(INX_OBJECTID) POS(3)
       INIT('GLOBALV                      ');
DCL DD INX_OBJECTID_AUT CHAR(2) DEF(INX_OBJECTID) POS(33)
       INIT(X'0000');

/*********************************************************************/
/* EXCEPTION MONITOR TO DETECT 2201X EXCEPTIONS (OBJECT NOT FOUND)   */
/*********************************************************************/

DCL EXCM EXCM_NOOBJECT EXCID(H"2201") INT(CREATE_INDEX) IMD;

/*********************************************************************/
/* PASA INVOCATION ENTRY FOR RETURN FROM EXCEPTION.                  */
/*********************************************************************/

DCL DD RTN_NOOBJECT CHAR(18) BDRY(16);
DCL SPCPTR RTN_NOOBJECT@ INIT(RTN_NOOBJECT);
DCL DD RTN_NOOBJECT_ADDR CHAR(16) DEF(RTN_NOOBJECT);
DCL DD RTN_NOOBJECT_OPT CHAR(1) DEF(RTN_NOOBJECT) POS(18)
       INIT(X'00');

/*********************************************************************/
/* RECEIVER VARIABLE FOR INDEPENDENT INDEX OPERATIONS.               */
/*********************************************************************/

DCL DD INX_RECEIVER CHAR(120);
DCL SPCPTR INX_RECEIVER@ INIT(INX_RECEIVER);

/*********************************************************************/
/* OPTION TEMPLATE FOR INDEPENDENT INDEX OPERATIONS.                 */
/*********************************************************************/

DCL DD INX_OPT CHAR(14);
DCL SPCPTR INX_OPT@ INIT(INX_OPT);
DCL SPC INX_OPT_SPC BAS(INX_OPT@);
DCL DD INX_OPT_RULE CHAR(2) DIR;
DCL DD INX_OPT_ARGL BIN(2) DIR;
DCL DD INX_OPT_ARGO BIN(2) DIR;
DCL DD INX_OPT_OCCC BIN(2) DIR;
DCL DD INX_OPT_RTNC BIN(2) DIR;
DCL DD INX_OPT_ELEN BIN(2) DIR;
DCL DD INX_OPT_EOFF BIN(2) DIR;

/*********************************************************************/
/* ARGUMENT VARIABLE FOR INDEPENDENT INDEX OPERATIONS.               */
/*********************************************************************/

DCL DD INX_ARG CHAR(120);
DCL SPCPTR INX_ARG@ INIT(INX_ARG);

/*********************************************************************/
/* START OF CODE                                                     */
/*********************************************************************/

  MATINVE RTN_NOOBJECT_ADDR,*,X'03'; /* MATERIALIZE THIS PROGRAM'S  */
                                     /* INVOCATION ENTRY IN THE     */
                                     /* PASA.  THIS ENTRY IS USED   */
                                     /* WHEN RETURNING FROM THE     */
                                     /* EXCEPTION HANDLER BELOW.    */

  RSLVSP INX@,INX_OBJECTID,*,*; /* RESOLVE TO "GLOBALV" USER INDEX  */
                                /* OBJECT.  IF THE OBJECT DOES NOT  */
                                /* EXIST, THEN THE X'2201' EXCEPTION*/
                                /* IS RETURNED, CAUSING THE "OBJECT */
                                /* NOT FOUND" EXCEPTION HANDLER AT  */
                                /* THE END OF THE PROGRAM TO RUN.   */



  CMPBLA(B) GV_REQUEST,'U'/NEQ(NOT_UPDATE); /* IF GV_REQUEST ¬= U   */
                                            /* BRANCH TO NOT_UPDATE */

    /* SET UP OPTIONS FOR INSERT INDEPENDENT INDEX ENTRY (INSINXEN) */
    /* OPERATION.                                                   */

    CPYBLA INX_OPT_RULE,X'0002';     /* RULE= INSERT.               */
    CPYNV  INX_OPT_OCCC,1;           /* OCCURENCE COUNT = 1.        */
    CPYBLA INX_ARG(1:10),GV_USERID;  /* SPECIFY INDEX ENTRY.        */
    CPYBLA INX_ARG(11:10),GV_VALUEID;
    CPYBLA INX_ARG(21:100),GV_VALUE;
    INSINXEN INX@,INX_ARG@,INX_OPT@; /* INSERT THE INDEX ENTRY.     */
    RTX *;                           /* RETURN                      */

NOT_UPDATE:
```

```
  CMPBLA(B) GV_REQUEST,'R'/NEQ(NOT_RETRIEVE); /* IF GV_REQUEST ¬= R */
                                              /* GOTO NOT_RETRIEVE. */

    /* SET UP OPTIONS FOR FIND INDEPENDENT INDEX ENTRY (FNDINXEN)   */
    /* OPERATION.                                                   */

    CPYBLA INX_OPT_RULE,X'0001';    /* RULE= FIND WITH EQUAL KEY.   */
    CPYNV  INX_OPT_ARGL,20;         /* ARGUMENT LENGTH= 20.         */
    CPYNV  INX_OPT_OCCC,1;          /* OCCURRENCE COUNT=1.          */
    CPYBLA INX_ARG(1:10),GV_USERID; /* SPECIFY SEARCH ARGUMENT.     */
    CPYBLA INX_ARG(11:10),GV_VALUEID;
    FNDINXEN INX_RECEIVER@,INX@,INX_OPT@,INX_ARG@; /* FIND ENTRY.   */
    CMPNV(B) INX_OPT_RTNC,1/EQ(FOUND_ENTRY); /* IF RETURN_COUNT = 1 */
                                             /* GOTO FOUND_ENTRY.   */
      CPYBLAP GV_VALUE,'*NONE',' '; /* ENTRY WAS NOT FOUND, SPECIFY */
                                    /* VALUE OF *NONE.              */
      RTX *;                        /* RETURN                       */

FOUND_ENTRY:

      CPYBLA GV_VALUE,INX_RECEIVER(21:100); /* ENTRY WAS FOUND,     */
                                            /* COPY VALUE TO USER   */
                                            /* PARAMETER.           */
      RTX *;                                /* RETURN               */

NOT_RETRIEVE:

  RTX *; /* UNKNOWN FUNCTION CODE.  RETURN.                         */

/*******************************************************************/
/* "OBJECT NOT FOUND" EXCEPTION HANDLER.                           */
/*******************************************************************/

ENTRY CREATE_INDEX INT;

  MODEXCPD EXCM_NOOBJECT,X'0000',X'01'; /* TURN OFF EXCEPTION       */
                                        /* MONITOR.                 */

  CALLX QUSCRTUI,QUSCRTUI_ARG,*; /* USE QUSCRTUI API TO CREATE THE  */
                                 /* USER INDEX OBJECT.              */

  RTNEXCP RTN_NOOBJECT@; /* RETURN FROM THE EXCEPTION HANDLER AND   */
                         /* RETRY THE OPERATION.                    */
PEND;
```

Creating a Batch Machine

These System C/400 PRPQ programs emulate a batch machine. One program, $USQEXSRV, acts as a server and takes the entries off a user queue and then runs the request through the Execute Command API, QCMDEXC. (The QCMDEXC API is described in the *CL Programmer's Guide*.) The other program, $USQEXREQ, acts as a requester and puts the entries into a user space. The APIs used in this example are:

- Create User Queue (QUSCRTUQ)
- Execute Command (QCMDEXC)

Requester Program ($USQEXREQ)

The following is a requester program using System C/400 PRPQ:

```
/*******************************************************************/
/*  PROGRAM:  $USQEXREQ                                            */
/*                                                                 */
/*  LANGUAGE:  SYSTEM C/400 PRPQ                                   */
/*                                                                 */
/*  DESCRIPTION:  This program enters commands to be run onto a    */
/*    user queue called TESTQ in library QGPL.  The user is        */
/*    prompted to enter as many commands (under 51 characters) as  */
/*    necessary.  To end the programs, enter QUIT at the prompt.   */
/*                                                                 */
/*  APIs USED:  None                                               */
/*                                                                 */
/*******************************************************************/
#define C400
#include <string.h>     /* strcpy, strncmp                         */
#include <miqueue.h>    /* _ENQ_msg_prefix, enq                    */
#include <miptrnam.h>   /* _SYSPTR, rslvsp                         */
#include "C400.h"       /* clibinit, clibterm, gets, printf */
                        /* _Usrq, _AUTH_ALL                        */


main()
{
_ENQ_msg_prefix e_msg_prefix;
_SYSPTR queue;
char INMsg[100];
/*******************************************************************/
/*  This resolves to the user queue created by $USQEXSRV.          */
/*******************************************************************/
queue = rslvsp(_Usrq,"TESTQ","QGPL",_AUTH_ALL);
e_msg_prefix.msg_len = 100;
/*******************************************************************/
/*  This establishes a linkage to a C/400 program which does I/O   */
/*  operations.                                                    */
/*******************************************************************/
clibinit;
/*******************************************************************/
/*  This loops until the user enters 'quit' as the command.        */
/*******************************************************************/
  while (1==1) {
    printf("\nEnter command to put on queue, or 'quit' \n ");
    gets(INMsg);
    printf("\nCommand entered was ==> %.100s\n",INMsg);
/*******************************************************************/
/*  This checks to see if the user entered 'quit' as the command.  */
/*  If it is true, the 'while' loop is ended.                      */
/*******************************************************************/
    if ((strncmp(INMsg,"quit",4) == 0)||(strncmp(INMsg,"QUIT",4) == 0))
      { break; }
/*******************************************************************/
/*  This adds the user-entered command to the user queue.          */
/*******************************************************************/
    enq(queue,&e_msg_prefix,INMsg);
    strcpy(INMsg," ");
  } /*while*/
/*******************************************************************/
/*  This adds the command END to the user queue which causes the   */
/*  end of the server program ($USQEXSRV).                         */
/*******************************************************************/
  strcpy(INMsg,"END");
  enq(queue,&e_msg_prefix,INMsg);
clibterm;
} /* $USQEXREQ */
```

To create the requester program using System C/400 PRPQ, specify the following:

```
CRTSYSCTXT TXT(QGPL/$USQEXREQ) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/$USQEXREQ) FROMTXT(QGPL/$USQEXREQ)
```

Server Program ($USQEXSRV)

The following is the server program using System C/400 PRPQ:

```
/*******************************************************************/
/*  PROGRAM:  $USQEXSRV                                            */
/*                                                                 */
/*  LANGUAGE:  System C/400 PRPQ                                   */
/*                                                                 */
/*  DESCRIPTION:  This program removes commands that are to be run */
/*    from a user queue called 'TESTQ' in library 'QGPL'.  The     */
/*    commands are removed and run in FIFO order.  The user queue  */
/*    is created before its use and should be deleted after you use*/
/*    this example program.  This program ends when it gets the    */
/*    command 'END' from the user queue.                           */
/*       The flow is as follows:                                   */
/*         (1)  Create the user queue.                             */
/*         (2)  Enter the loop                                     */
/*         (3)     Wait forever for a command on the user queue.   */
/*         (4)      If the command END exists, end the loop.       */
/*         (5)      Or else run the command and restart the loop.  */
/*         (6)  End the loop                                       */
/*    For the best results, the user should call this program and  */
```

```
/*    call $USQEXREQ from another session.                          */
/*                                                                  */
/* APIs USED:  QCMDEXC, QUSCRTUQ                                    */
/*                                                                  */
/******************************************************************/
#define C400
#include <string.h>      /* strlen, strncmp           */
#include <micomput.h>    /* _DPA_Template_T, cvtnci   */
#include <miqueue.h>     /* _DEQ_msg_prefix, deq      */
#include <miptrnam.h>    /* _SYSPTR, rslvsp           */
#include "OPUSAPI.h"     /* Linkage info, structures  */
#include "C400.h"        /* _Usrq, _AUTH_ALL          */


main()
{

/******************************************************************/
/*    This sets up the interface to the external QCMDEXC API      */
/*    to run the commands extracted from the user queue.          */
/******************************************************************/
#pragma linkage(QCMDEXC,OS)
  void QCMDEXC (char comm_to_exec[],
                char length_of_call[15],
                char third_param[3]);

_DEQ_msg_prefix d_msg_prefix;
_DPA_Template_T d_tmp_t;
_SYSPTR queue;
char OUTMsg[100];
int cmd_name_lngth;
char pack_name_lngth[15];
char igc_param[] = "IGC";
/******************************************************************/
/*    This sets up the parameters to call to the QUSCRTUQ API.    */
/******************************************************************/
char q_name[]= "TESTQ     QGPL      ";
char ext_atr[]= "TESTER    ";
char q_type[]= "F";
int key_lngth = 0;
int max_msg_s = 100;
int int_msgs  = 10;
int add_msgs = 50;
char auth[]  = "*ALL      ";
char desc[] = "Description .....    ";
/******************************************************************/
/*  This calls the QUSCRTUQ API to create the user queue.         */
/******************************************************************/
  QUSCRTUQ(q_name,ext_atr,q_type,key_lngth,max_msg_s,int_msgs,
           add_msgs,auth,desc);
/******************************************************************/
/*  This resolves to the created user queue.                      */
/******************************************************************/
  queue = rslvsp(_Usrq,"TESTQ","QGPL",_AUTH_ALL);
/******************************************************************/
/*  This sets the dequeue operation to wait for the command forever.*/
/******************************************************************/
  d_msg_prefix.wait_forever = 1;
  d_tmp_t.Type = T_Packed;
  d_tmp_t.Length = 0x050F;
  d_tmp_t.reserved = 0x00000000;
/******************************************************************/
/*  This loops until the command 'END' is removed from the user   */
/*  queue.                                                        */
/******************************************************************/
  while (1==1) {
    deq(&d_msg_prefix,OUTMsg,queue);
/******************************************************************/
/*  This checks to see if the command 'END' is removed.  If this is */
/*  true, then end the 'while' loop.                              */
/******************************************************************/
    if (strncmp(OUTMsg,"END",3) == 0)
      { break; }
    cmd_name_lngth = strlen(OUTMsg);
/******************************************************************/
/*  This converts the integer in cmd_name_lngth to a packed decimal */
/*  using arguments in d_tmp_t.                                   */
/******************************************************************/
    cvtnci(pack_name_lngth,cmd_name_lngth,d_tmp_t);
/******************************************************************/
/*  This runs the command removed from the user queue.            */
/******************************************************************/
    QCMDEXC(OUTMsg,pack_name_lngth,igc_param);
  } /* while */
}  /* $USQEXSRV */
```

To create the server program using System C/400 PRPQ, specify the following:

```
CRTSYSCTXT TXT(QGPL/$USQEXSRV) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/$USQEXSRV) FROMTXT(QGPL/$USQEXSRV)
```

Using the Create Program (QPRCRTPG) API

The following PL/I program shows how the QPRCRTPG API can be used to create a program:

```
/******************************************************************/
/*                                                                */
/* MODULE: QPRUSRP                                                */
/*                                                                */
/* LANGUAGE:  PL/I                                                */
/*                                                                */
/* FUNCTION: Creates a program using the QPRCRTPG API.  Reads the */
/*           intermediate representation of program to be used from */
/*           a source file member.                                */
/*                                                                */
/* APIs used: QPRCRTPG                                            */
/*                                                                */
/******************************************************************/
QPRUSRP: PROCEDURE (PGM, TEXT, SRCF, SRCM, SRCDT, AUT, OPTIONS);

DECLARE
  PGM CHAR(20),                 /* The first ten characters       */
                                /* specifies the name of the      */
                                /* program to be created.  The    */
                                /* second ten characters specifies */
                                /* the library to be used.        */

  TEXT CHAR(50),                /* Contains a brief description of */
                                /* the program.                   */

  SRCF CHAR(20),                /* The first ten characters       */
                                /* specify the name of the source */
                                /* file containing the intermediate*/
                                /* representation of program. The */
                                /* second ten characters specify  */
                                /* the library containing the     */
                                /* source file.                   */

  SRCM CHAR(10),                /* Specifies the source file      */
                                /* member containing the          */
                                /* intermediate representation of */
                                /* program.                       */

  SRCDT CHAR(13),               /* Specifies the last changed date */
                                /* and time for the source file   */
                                /* member.                        */

  AUT CHAR(10),                 /* Specifies the authority to be  */
                                /* used.                          */

  OPTIONS CHAR(176);            /* Specifies the option values to */
                                /* be used.  These are stored as a */
                                /* series of CHAR(11) values.     */

DECLARE
  IRPRGM FILE;                  /* The source file containing the */
                                /* intermediate representation of */
                                /* program.                       */

DECLARE
  QPRCRTPG ENTRY(*, *, *, *, *, *, *, *, *, *, *, *, *)
           OPTIONS(ASSEMBLER);
                                /* This declares the QPRCRTPG API. */

DECLARE
  INPUT_BUFFER CHAR(92),        /* Used to read data from the     */
                                /* file.  The first 12 bytes      */
                                /* contain the time/date stamp.   */
                                /* The remaining 80 bytes contain */
                                /* source data.                   */

  INPUT_ARRAY(32000) CHAR(80),  /* Used to store the intermediate */
```

```
                              /* representation of program.  Up  */
                              /* to 32000 records may be         */
                              /* processed.                      */

   INPUT_SIZE BINARY(31),     /* Contains the number of input    */
                              /* records processed.              */

   PAGE_NUM BINARY(31) STATIC INIT(1),
                              /* Specifies the starting page     */
                              /* number.                         */

   NUM_OPTIONS BINARY(31) STATIC INIT(16);
                              /* Specifies the number of option  */
                              /* values in the option template.  */

DECLARE
  1 BIT_FLAGS STATIC,
    3 MORE_RECORDS BIT(1) ALIGNED,
    3 NO           BIT(1) ALIGNED INIT('0'),
    3 YES          BIT(1) ALIGNED INIT('1');

ON ENDFILE(IRPRGM)
  MORE_RECORDS= NO;

  OPEN FILE(IRPRGM);

  MORE_RECORDS= YES;
  INPUT_SIZE= 0;
  GET FILE(IRPRGM) EDIT(INPUT_BUFFER)(A(92));
  DO WHILE(MORE_RECORDS);
    INPUT_SIZE= INPUT_SIZE + 1;
    INPUT_ARRAY(INPUT_SIZE)= SUBSTR(INPUT_BUFFER, 13, 80);
    GET FILE(IRPRGM) EDIT(INPUT_BUFFER)(A(92));
  END; /* DO WHILE */

  IF INPUT_SIZE > 0 THEN
  DO;
    CALL QPRCRTPG (INPUT_ARRAY
                  ,INPUT_SIZE * 81
                  ,PGM
                  ,TEXT
                  ,SRCF
                  ,SRCM
                  ,SRCDT
                  ,'QSYSPRT   *LIBL     '
                  ,PAGE_NUM
                  ,AUT
                  ,OPTIONS
                  ,NUM_OPTIONS
                  );
  END; /* IF */

  CLOSE FILE(IRPRGM);
END QPRUSRP;
```

Using Profile Handles

The following example illustrates how to generate, change, and release profile handles in a CL program. The example uses three of the security APIs:

- Get Profile Handle (QSYGETPH)
- Set Profile (QWTSETP)
- Release Profile Handle (QSYRLSPH)

```
PGM (&USERID &PWD)

/* Declare the variables needed by this program:           */
DCL        VAR(&USERID) TYPE(*CHAR) LEN(10)
DCL        VAR(&PWD) TYPE(*CHAR) LEN(10)
DCL        VAR(&SECOFR) TYPE(*CHAR) LEN(10) VALUE(QSECOFR)
DCL        VAR(&SECPWD) TYPE(*CHAR) LEN(10) VALUE(*NOPWD)
DCL        VAR(&PRFHNDL1) TYPE(*CHAR) LEN(12)
DCL        VAR(&PRFHNDL2) TYPE(*CHAR) LEN(12)

/* Generate profile handles for the QSECOFR user ID and    */
/* for the user ID passed to this program:                 */
CALL       PGM(QSYGETPH) PARM(&SECOFR &SECPWD &PRFHNDL1)
CALL       PGM(QSYGETPH) PARM(&USERID &PWD &PRFHNDL2)

/* Change the user for this job to the user ID passed to   */
/* this program:                                           */
CALL       PGM(QWTSETP) PARM(&PRFHNDL2)

/* This program is now running under the user ID passed to */
/* this program.

/* Now change the user ID for this job back to the QSECOFR */
/* user ID:                                                */
CALL       PGM(QWTSETP) PARM(&PRFHNDL1)

/* The profile handles generated in this program can now   */
/* be released:                                            */
CALL       PGM(QSYRLSPH) PARM(&PRFHNDL1)
CALL       PGM(QSYRLSPH) PARM(&PRFHNDL2)

ENDPGM
```

Generating and Sending an Alert

The following Pascal program uses both alert APIs. First, it calls the Generate Alert API, QALGENA, to generate an alert without sending a message to the QSYSOPR or QHST message queue. Then it uses the Send Alert API, QALSNDA, to send the alert to the OS/400 alert manager for further processing.

```
program example;

type
  (* ************************************ *)
  (* Simple string types.                 *)
  (* ************************************ *)
  string07  = packed array(.1..7.)   of char;
  string10  = packed array(.1..10.)  of char;
  string512 = packed array(.1..512.) of char;
  string100 = packed array(.1..100.) of char;

  (* ************************************ *)
  (* Alert table record.  This is also the *)
  (* message file for the message ID.     *)
  (* ************************************ *)
  alrtbl = packed record
    objname : string10;
    libname : string10;
  end;

  (* ************************************ *)
  (* Error code  structure.  The BYTESPROV *)
  (* field is input; the rest are output.  *)
  (* ************************************ *)
  errorcode = packed record
    bytesprov     : integer;
    bytesavail    : integer;
    messageid     : string07;
    reserved      : char;
    exceptiondata : string100;
  end;

(* ************************************************** *)
(* This is the application program interface to create an *)
(* alert when given a message ID and alert table.     *)
(*                                                    *)
(* The parameters passed are:                         *)
(*    ALERT      - O   - The  alert  major vector that is *)
(*                       created                      *)
(*    PROVIDED   - I   - The number of bytes of  provided *)
(*                       space                        *)
(*    NEEDED     - O   - The  number  of  bytes of needed *)
(*                       space                        *)
(*    ALERTTBL   - I   - The alert table and message file *)
(*                       containing the alert description *)
(*                       and message description      *)
(*    MESSAGE    - I   - The message ID to use        *)
```

```
(*    MSGDATA    - I   - The message substitution text *)
(*    MSGDATALEN - I   - The number of bytes provided in *)
(*                       MSGDATA                        *)
(*    RETURNCODE - I/O - The error code structure       *)
(* ************************************************** *)
procedure qalgena(
   var alert      : string512;
   var provided   : integer;
   var needed     : integer;
   var alerttbl   : alrtbl;
   var message    : string07;
   var msgdata    : string100;
   var msgdatalen : integer;
   var returncode : errorcode); nonpascal;

(* ************************************************** *)
(* This is the application program interface to send an  *)
(* alert to the alert manager for further processing.    *)
(*                                                       *)
(* The parameters passed are:                            *)
(*    ALERT      - I   - The alert major vector that is  *)
(*                       sent                            *)
(*    PROVIDED   - I   - Number of bytes provided by the *)
(*                       alert major vector              *)
(*    LCLRCVIND  - I   - An indicator of a local or      *)
(*                       received alert                  *)
(*    ORIGIN     - I   - The origin of the alert         *)
(*    RETURNCODE - I/O - The error code structure        *)
(* ************************************************** *)
procedure qalsnda(
   var alert      : string512;
   var provided   : integer;
   var lclrcvind  : char;
   var origin     : string10;
   var returncode : errorcode); nonpascal;

var
  (* ********************************** *)
  (* Local copies of the parameters     *)
  (* ********************************** *)
   alert     : string512;         (* The alert major vector space    *)
   bytespass : integer;           (* The number of bytes passed in   *)
   bytesrtn  : integer;           (* The number of bytes returned    *)
   table     : alrtbl;            (* The alert table/message file    *)
   msgid     : string07;          (* The message ID to use           *)
   msgdata   : string100;         (* The message data to substitute  *)
   msglength : integer;           (* The length of the message data  *)
   localrecd : char;              (* The local/received indicator    *)
   origin    : string10;          (* The origin of the alert to send *)
   rtncode   : errorcode;         (* The return code for the APIs    *)

begin

   (* Start by generating an alert for a specific message *)
   alert             := ' ';              (* Initialize alert space   *)
   bytespass         := length(alert);    (* Initialize in length     *)
   bytesrtn          := 0;                (* Initialize out length    *)
   msgid             := 'CPA2601';        (* Initialize message ID    *)
   table.objname     := 'QCPFMSG   ';     (* Initialize table name    *)
   table.libname     := 'QSYS      ';     (* Initialize table library *)
   msgdata           := ' ';              (* Initialize data          *)
   msglength         := 0;                (* Initialize data length   *)
   rtncode.bytesprov := 116;              (* Initialize RC length     *)
   rtncode.messageid := '       ';       (* Initialize RC message    *)
   qalgena(alert,
           bytespass,
           bytesrtn,
           table,
           msgid,
           msgdata,
           msglength,
           rtncode);                      (* Generate the alert       *)

   (* Check the return code from QALGENA *)
   if rtncode.messageid = '       ' then
      (* Return code is OK, so send the generated alert *)
      begin                                        (* Alert OK         *)
         localrecd             := 'L';             (* Local alert      *)
         origin                := 'EXAMPLE   ';    (* From EXAMPLE pgm *)
         rtncode.bytesprov     := 116;             (* Init RC length   *)
         rtncode.messageid     := '       ';       (* Blank RC message *)
         rtncode.exceptiondata := ' ';             (* Blank RC data    *)
         qalsnda(alert,
                 bytesrtn,
                 localrecd,
                 origin,
                 rtncode);                   (* Send the alert    *)
      end                                    (* ...Alert OK       *)
   else
      (* QALGENA failed, so print out the exception ID *)
      begin                                  (* QALGENA failed    *)
         writeln('QALGENA had an error');    (* Tell user about   *)
         write('The error was ');            (*  the error and    *)
         writeln(rtncode.messageid);         (*  the exception ID *)
      end;                                   (* ...QALGENA failed *)

end.
```

Diagnostic Reporting

The following example program illustrates the use of the Send Nonprogram Message API, QMHSNDM, and the Receive Program Message API, QMHRCVPM. The program produces a diagnostic report of errors that occur when the QMHSNDM API is used to send a message to more than one message queue.

The program calls the QMHSNDM API to send a message to message queues that do not exist. The QMHSNDM API returns a generic exception message, CPF2469. This message indicates that the API also returned one or more diagnostic messages describing the errors. After the program receives the exception message and verifies that it is message CPF2469, it uses the QMHRCVPM API to receive the diagnostic messages. The program prints the exception message, the diagnostic messages, and the message help.

Diagnostic Report (DIAGRPT) Program

```
/*****************************************************************/
/*                                                               */
/* MODULE NAME:  DIAGRPT - Diagnostic Report                     */
/* LANGUAGE:     C/400                                           */
/*                                                               */
/* FUNCTION:     This module will produce a diagnostic report that */
/*               could be used in diagnosing the errors that     */
/*               occurred using the QMHSNDM API to send a message */
/*               to multiple message queues.                     */
/*               This program purposely causes the QMHSNDM API to */
/*               try to send a message to message queues that do not*/
/*               exist and as a result the generic CPF2469 exception*/
/*               is returned indicating that 1 or more diagnostic */
/*               messages were returned identifying the error(s) on */
/*               the send.  The program receives the exception and */
/*               prints it out, and if it is the CPF2469, it     */
/*               receives the previous diagnostics and also prints */
/*               them out.                                       */
/*                                                               */
/* Dependency:   A print file must be created before calling program*/
/*               DIAGRPT.  The print file should be created using */
/*               the following command-                          */
/*               CRTPRTF FILE(PRTDIAG) CTLCHAR(*FCFC) CHLVAL(1 (13))*/
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <psmhapi.h>

#define  DIAG_TYPE "02"
#define  BUF_SIZE  80

/*****************************************************************/
/* Type definition for error code structure                      */
/*****************************************************************/
typedef struct error_code_struct
  {
```

```
   int bytes_provided;
   int bytes_available;
   char exception[7];
   char RESERVED;
   char exception_data[100];
  } error_code_struct;

/*********************************************************************/
/* Type definition for qualified name structure                      */
/*********************************************************************/
typedef struct qual_name_struct
  {
   char  name[10];
   char  libr[10];
  } qual_name_struct;

/*********************************************************************/
/* Type definition for message information structure used on the     */
/* receive.  F is the fixed portion of the record  and V is the      */
/* variable length portion of the record.                            */
/*********************************************************************/
typedef struct msg_info_struct
  {
   RCVM0200     F;
   char         V[1200];
  } msg_info_struct;

FILE  *prtf;
char  buf[80];
char  received[7];
int   exception_count;


/*********************************************************************/
/* Function to handle errors received on the API calls.              */
/*********************************************************************/
 void error_handler(void)
 {
   sigdata_t *data=sigdata();

   data->sigact->xhalt=0;
   data->sigact->xpmsg=0;
   data->sigact->xumsg=0;
   data->sigact->xdebug=0;
   data->sigact->xdecerr=0;
   data->sigact->xresigprior=0;
   data->sigact->xresigouter=0;
   data->sigact->xremovemsg=0;
   data->sigact->xrtntosgnler=0;
   memcpy(received,data->exmsg->exmsgid,7);
   exception_count++;
   signal(SIGABRT,error_handler);
 }


/*********************************************************************/
/* BuildQList:  Routine to build the message queue list.             */
/*********************************************************************/
void BuildQList(QueueList,NumQueue)

  qual_name_struct *QueueList;
  int              NumQueue;
{
  int i;

  strncpy(QueueList[0].name,"QPGMR     ",10);
  strncpy(QueueList[1].name,"SNOOPY    ",10);
  strncpy(QueueList[2].name,"QSECOFR   ",10);
  strncpy(QueueList[3].name,"PEANUTS   ",10);
  strncpy(QueueList[4].name,"QUSER     ",10);

  for (i = 0; i < NumQueue ; i++ )
    {
     strncpy(QueueList[i].libr,"*LIBL     ",10);
    }
}


/*********************************************************************/
/* PrintError: Routine to print error information & exit.            */
/*********************************************************************/
void PrintError(errstring,exception)
  char *errstring;
  char  exception[7];
{

  memset(buf,' ',BUF_SIZE);
  buf[0] = '0';
  strncpy(buf+1,errstring,strlen(errstring));
  fwrite(buf,1,BUF_SIZE,prtf);

  memset(buf,' ',BUF_SIZE);
  buf[0] = '0';
  strncpy(buf+1,"Exception received->",14);
  strncpy(buf+15,exception,strlen(exception));
  fwrite(buf,1,BUF_SIZE,prtf);
  fclose(prtf);
  exit(1);
 }


/*********************************************************************/
/* PrintData: Routine to print varying length character string data.*/
/*********************************************************************/
void PrintData(strname,strptr,strlgth)
char *strname;
void *strptr;
int strlgth;
{
  char *strdata = strptr;
  int i,lgth,remain;

  /* Write the description and the data that will fit on one line   */
  memset(buf,' ',BUF_SIZE);
  buf[0] = '0';
  lgth = strlen(strname);
  strncpy(buf+1,strname,lgth);
  lgth++;

  /* remain = MIN(strlgth,80 - lgth)  */
  remain = (strlgth < 80 - lgth) ? strlgth : 80 - lgth;
  strncpy(buf+lgth,strdata,remain);
  fwrite(buf,1,BUF_SIZE,prtf);

  /* Now write the remainder of the data */
  if (strlgth > (80 - lgth ))
    {
     /* Adjust pointer to data not printed yet */
     strdata = strdata + (80 - lgth);

     for (i = 0; i < strlgth; i = i + 70, strdata = strdata + 70 )
       {
        /* lgth = MIN(strlgth-i,70)  */
        lgth = (strlgth-i < 70) ? strlgth-i : 70;

        memset(buf,' ',BUF_SIZE);
        strncpy(buf,"0         ",10);
        memcpy(buf+10,strdata,lgth);
        fwrite(buf,1,BUF_SIZE,prtf);
       }

    }
}


/*********************************************************************/
/* PrintMessage: Routine to print the message data & text.           */
/*********************************************************************/
void PrintMessage(Msg)

  msg_info_struct *Msg;

{
  char *DataPtr;     /* Pointer to the varying length character data*/
  int  DataLen;      /* Length of the varying length character data */
  char CharType[10]; /* Message type as a character string          */

  PrintData("Message ID->",Msg->F.msg_id,7);
  /* Convert Message Type to a character string to be printed out   */
  if (memcmp(Msg->F.msg_type,"02",2)==0)
    strncpy(CharType,"DIAGNOSTIC", 10);
  else if (memcmp(Msg->F.msg_type,"15",2)==0)
    strncpy(CharType,"ESCAPE    ",10);
  PrintData("Message Type->",CharType,10);

  /* First point to the beginning of the message data               */
```

```
  /* in the structure and get the length of data returned.          */
  DataPtr = Msg->V;
  DataLen = Msg->F.data_returned;
  /* If there is non-blank data, print it out                       */
  if ((DataLen > 0) && (strspn(DataPtr," ") < DataLen))
    PrintData("Message data received->",DataPtr,DataLen);

  /* Point to the beginning of the message text field and get the   */
  /* length of message text returned.                               */
  DataPtr += DataLen;
  DataLen = Msg->F.text1_returned;
  /* If there is non-blank text, print it out                       */
  if ((DataLen > 0) && (strspn( DataPtr," ") < DataLen))
    PrintData("Message text received->",DataPtr,DataLen);

  /* Now update to point to the beginning of the message            */
  /* help text field and get the length of message help text        */
  /* returned.                                                      */
  DataPtr += DataLen;
  DataLen = Msg->F.text2_returned;
  /* If there is non-blank message help text, print it out          */
  if ((DataLen > 0) && (strspn( DataPtr," ") < DataLen))
    PrintData("Message help text received->",DataPtr,DataLen);
  strncpy(buf,"-                                                ",43);
  fwrite(buf,1,BUF_SIZE,prtf);
}

 /*********************************************************************/
 /*                                                                   */
 /*  Start of main program.                                           */
 /*                                                                   */
 /*********************************************************************/

main()
{


 error_code_struct ErrorCode;

 qual_name_struct  MsgQList[5];
 qual_name_struct  MsgFile;
 qual_name_struct  RpyMsgQ;

 msg_info_struct   MsgInfo;

 char MsgData[128];
 char MsgText[512];
 char MsgHelp[512];
 char PgmMsgQ[10];
 char MsgType[10];
 char MsgAction[10];
 char Format[8];
 char MsgId[7];
 char MsgKey[4];

 int  MsgTextLen;
 int  MsgInfoLen;
 int  NumMsgQ;
 int  PgmCount;
 int  WaitTime;
 int  morediag;


 /* Initialize variables */
 exception_count = 0;
 signal(SIGABRT,error_handler);
 memcpy(ErrorCode.exception,"       ",7);
 ErrorCode.bytes_provided = 0;

 memcpy(MsgId,"       ",10);
 memcpy(MsgFile.name,"          ",10);
 memcpy(MsgFile.libr,"          ",10);
 strcpy(MsgText,"This is an immediate, informational message");
 MsgTextLen = strlen(MsgText);
 memcpy(MsgType,"*INFO     ",10);
 memcpy(RpyMsgQ.name,"          ",10);
 memcpy(RpyMsgQ.libr,"          ",10);

 /* Build the list of message queues to send the message to        */
 NumMsgQ = 5;
 BuildQList(MsgQList,NumMsgQ);

 /* Send the message to the list of message queues.                */
 QMHSNDM( MsgId,
         &MsgFile,
          MsgText,
          MsgTextLen,
          MsgType,
         &MsgQList,
          NumMsgQ,
         &RpyMsgQ,
         &MsgKey,
         &ErrorCode);

 /* If an error occurred on the send, produce an exception report  */
 /* identifying what errors occurred.                              */
 if (exception_count != 0)
   {

    /* Open printer file using first character forms control & write*/
    /* the header information.                                     */
    prtf = fopen ("PRTDIAG", "wb type=record recfm=FA lrecl=80");
    memset(buf,' ',BUF_SIZE);
    strncpy(buf,"1                          DIAGNOSTIC REPORT",43);
    fwrite(buf,1,BUF_SIZE,prtf);
    strncpy(buf,"                          -----------------",43);
    fwrite(buf,1,BUF_SIZE,prtf);
    strncpy(buf,"-                                           ",43);
    fwrite(buf,1,BUF_SIZE,prtf);

    /* Do the setup to first receive the exception signalled.      */
    memcpy(Format,"RCVM0200",8);
    memcpy(PgmMsgQ,"*         ",10);
    memcpy(MsgType,"*EXCP     ",10);
    memcpy(MsgKey,"    ",4);
    memcpy(MsgAction,"*OLD      ",10);
    PgmCount = 0;
    WaitTime = 0;
    MsgInfoLen = 1276;

    /* Now change bytes_provided to 116 so that if any errors occur */
    /* on the receive, the error information will be returned in the*/
    /* error code structure instead of generating more exceptions   */
    /* which will clutter up the program message queue.             */
    ErrorCode.bytes_provided = 116;

    /* Receive the last exception type message on the program      */
    /* message queue                                               */
    QMHRCVPM(&MsgInfo,
             MsgInfoLen,
             Format,
             PgmMsgQ,
             PgmCount,
             MsgType,
             MsgKey,
             WaitTime,
             MsgAction,
            &ErrorCode);

    /* Test for any errors on the receive */
    if (ErrorCode.bytes_available > 0)
      {
       PrintError("QMHRCVPM - Did not complete successfully",
                  ErrorCode.exception);
      }

    /* An exception message was received successfully.  Now see if  */
    /* the message received is the same exception that was signalled*/
    /* If not, there is an error.                                   */
    if (strncmp(MsgInfo.F.msg_id,received,7) != 0)
      {
       PrintError("QMHRCVPM - Wrong exception received",
                  MsgInfo.F.msg_id);
      }


    /* The exception message was received successfully.            */
    /* Print the message data & text for the exception message.    */
    PrintMessage(&MsgInfo);

    /* If the message was the generic CPF2469, there are 1 or more  */
    /* diagnostic messages to go with the CPF2469 on the queue.     */
    /* Receive the diagnostic messages previous to the CPF2469 until*/
    /* a non-diagnostic message is received or there are no more    */
    /* messages.                                                    */
```

```
    if (strncmp(MsgInfo.F.msg_id,"CPF2469",7) == 0)
      {
       memcpy(MsgType,"*PRV      ",10);
       memcpy(MsgKey,MsgInfo.F.msg_key,4);
       morediag = 1;

       while(morediag == 1)
         {
          /* Receive the previous diagnostic */
          QMHRCVPM(&MsgInfo,
                   MsgInfoLen,
                   Format,
                   PgmMsgQ,
                   PgmCount,
                   MsgType,
                   MsgKey,
                   WaitTime,
                   MsgAction,
                   &ErrorCode);

          /* Test for error on the receive */
          if (ErrorCode.bytes_available > 0)
            {
             PrintError("QMHRCVPM - Did not complete successfully",
                        ErrorCode.exception);
            }

          /* If bytes available = 0 OR the next message is not a    */
          /* diagnostic message, we are done.                       */
          if ((MsgInfo.F.bytes_available == 0) ||
              (strncmp(MsgInfo.F.msg_type,DIAG_TYPE,2) != 0) )
            {
             morediag = 0;
            }
          else                /* A diagnostic was received          */
            {
             /* Print the message data & text for the diagnostic msg*/
             PrintMessage(&MsgInfo);

             /* Now copy the message key of the diagnostic message  */
             /* received to the MsgKey parameter to use on the next */
             /* call to QMHRCVPM.                                   */
             memcpy(MsgKey,MsgInfo.F.msg_key,7);
            }

         }                    /* End of while morediag = 1          */

      }                       /* End of if CPF2469 received         */

    /* Write trailer */
    memset(buf,' ',BUF_SIZE);
    strncpy(buf,"-                    END OF DIAGNOSTIC REPORT",48);
    fwrite(buf,1,BUF_SIZE,prtf);

    /* Close the print file */
    fclose(prtf);

   }                          /* End of if error on send            */

}                             /* End mainline                       */
```

Printed Diagnostic Report

The DIAGRPT program produces a report like this:

```
Message ID->CPF2469
Message Type->ESCAPE
Message text received->Error occurred when sending message.
Message help text received->Recovery  . . . :   See messages
         previously listed for a description of the error.
         Correct the error, and then try the
         command again.

Message ID->CPF2403
Message Type->DIAGNOSTIC
Message data received->WILLIAM   *LIBL
Message text received->Message queue WILLIAM in *LIBL not found.
Message help text received->Cause . . . . . :   The message queue you
        specified was not found in the library you specified.  One
        of the following occurred: -- The queue name was not
        entered correctly. -- The queue does not exist in the
        specified library. -- You specified the wrong library name.
        Recovery  . . . :   Do one of the following and try the
        request again: -- Correct or change the message queue
        name or library name used in the message queue (MSGQ)
        parameter or the to-message queue (TOMSGQ) parameter.
        -- Create the message queue using the Create Message
        Queue (CRTMSGQ) command.


Message ID->CPF2403
Message Type->DIAGNOSTIC
Message data received->ANDREW    *LIBL
Message text received->Message queue ANDREW in *LIBL not found.
Message help text received->Cause . . . . . :   The message queue
         you specified was not found in the library you specified.
         One of the following occurred: -- The queue name was not
         entered correctly. -- The queue does not exist in the
         specified library. -- You specified the wrong library
         name. Recovery  . . . :   Do one of the following and
         try the request again: -- Correct or change the message
         queue name or library name used in the message queue
         (MSGQ) parameter or the to-message queue (TOMSGQ)
         parameter. -- Create the message queue using the Create
         Message Queue (CRTMSGQ) command.
```

Listing Directories

To call this program, enter

```
'CALL DIR PARM(/'QDLS')'
```

```
/*********************************************************************/
/*********************************************************************/
/*  FUNCTION:  This program lists a directory to a spooled file.     */
/*             The list resembles the DOS DIR command output.        */
/*                                                                   */
/*  LANGUAGE:  PL/I                                                  */
/*                                                                   */
/*  APIs USED: QHFOPNDR, QHFRDDR, QHFCLODR, QHFLSTFS, QUSCRTUS,      */
/*             QUSRTVUS                                              */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
DIR: PROCEDURE(Dir_Name) OPTIONS(MAIN);

/* parameter declarations                                            */
DCL Dir_Name CHARACTER (100);

/* API entry declarations                                            */
/*                                                                   */
/* The last parameter, the error code, is declared as FIXED BIN(31) */
/* for all APIs.  This always has a value of zero, specifying that  */
/* exceptions should be returned.                                    */
DCL QHFOPNDR ENTRY(CHAR(16),CHAR(100),FIXED BIN(31),CHAR(6),
                   CHAR(52),FIXED BIN(31),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

DCL QHFCLODR ENTRY(CHAR(16),FIXED BIN(31)) OPTIONS(ASSEMBLER);

DCL QHFRDDR ENTRY(CHAR(16),CHAR(200),FIXED BIN(31),FIXED BIN(31),
                  FIXED BIN(31),FIXED BIN(31),FIXED BIN(31))
                  OPTIONS(ASSEMBLER);

DCL QHFLSTFS ENTRY(CHAR (20),CHAR (8),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

DCL QUSCRTUS ENTRY(CHAR (20),CHAR (10),FIXED BIN(31),CHAR (1),
                   CHAR (10),CHAR (50),CHAR (10),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

DCL QUSRTVUS ENTRY(CHAR (20),FIXED BIN(31),FIXED BIN(31),CHAR(*),
                   FIXED BIN(31)) OPTIONS(ASSEMBLER);

DCL QWCCVTDT ENTRY(CHAR (10),CHAR (8),CHAR(10),CHAR(16),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

/*********************************************************************/
/* Parameters for QHFOPNDR                                           */
/*********************************************************************/
/* Directory handle                                                  */
  DCL Dir_Handle CHAR(16);
/* The path name is the Dir_name parameter passed into the           */
```

```
/* program.                                                    */
/*                                                             */
/* Length of path name                                         */
  DCL namelen FIXED BIN(31);
/* Open information                                            */
  DCL openinfo CHAR(6);
/* Attribute selection table                                   */
  DCL selection CHARACTER(52);           /* selection is used to */
  DCL 1 selection_Struct BASED(Sel_ptr), /* allocate space for   */
        2 num_att   FIXED BIN(31),       /* selction_Struct by   */
        2 offset1   FIXED BIN(31),       /* setting Sel_ptr to   */
        2 offset2   FIXED BIN(31),       /* the address of       */
        2 offset3   FIXED BIN(31),       /* selection.           */
        2 att_len1  FIXED BIN(31),
        2 att_name1 CHAR(8),
        2 att_len2  FIXED BIN(31),
        2 att_name2 CHAR(8),
        2 att_len3  FIXED BIN(31),
        2 att_name3 CHAR(8);
  DCL Sel_ptr POINTER;
/* Length of attribute selection table                      */
  DCL selectionlen FIXED BIN(31);
/* Error Code                                               */
  DCL Error_Code FIXED BIN(31);

/************************************************************/
/* Parameters for QHFRDDR                                   */
/************************************************************/
/* The directory handle is the same as for QHFOPNDR.        */
/*                                                          */
/* Data buffer                                              */
  DCL buffer CHARACTER(200);             /* buffer is used to    */
  DCL 1 buffer_Struct BASED(Buf_ptr),    /* allocate space for   */
        2 num_dir_ret FIXED BIN(31),     /* buffer_Struct this is */
        2 offset_dir  FIXED BIN(31),     /* done by assigning the */
        2 num_att     FIXED BIN(31),     /* address of buffer to  */
        2 offsets(4)  FIXED BIN(31),     /* Buf_ptr.              */
        2 attinfo     CHARACTER(276);
  DCL Buf_ptr POINTER;
/* The length of the data buffer and number of directory entries */
/* to retrieve are hard coded into the call to QHFRDDR.     */
/*                                                          */
/* Number of directory entries retrieved                    */
  DCL result_count FIXED BIN(31);
/* Length of data returned                                  */
  DCL bytes_returned FIXED BIN(31);
/* The error code is the same as used for QHFOPNDR.         */
/************************************************************/

/************************************************************/
/* Parameters for QHFCLODR                                  */
/************************************************************/
/* The directory handle and error code are the same as used in */
/* the QHFOPNDR and QHFRDDR APIs.                           */
/************************************************************/

/************************************************************/
/* Parameters for QUSCRTUS                                  */
/************************************************************/
/* The qualified user space name is hard coded into the call to */
/* QUSCRTUS.                                                */
/*                                                          */
/* The extended attribute is hard coded into the call.      */
/*                                                          */
/* Initial size                                             */
  DCL size FIXED BIN(31);
/* The initial value is hard coded into the call.           */
/*                                                          */
/* The public authority is hard coded into the call.        */
/*                                                          */
/* Text description                                         */
  DCL text CHARACTER(50);
/* The replacement option is hard coded into the call.      */
/*                                                          */
/* The error code is the same as for the above APIs.        */
/************************************************************/

/************************************************************/
/* Parameters for QHFLSTFS                                  */
/************************************************************/
/* The qualified user space name and format name are hard coded */
/* into the call to QHFLSTFS.                               */
/* The error code is the same as for the above APIs.        */
/************************************************************/

/************************************************************/

/************************************************************/
/* Parameters for QUSRTVUS                                  */
/************************************************************/
/*                                                          */
/* The qualified user space name is hard coded into the call. */
/*                                                          */
/* Starting position                                        */
  DCL startpos FIXED BIN(31);
/* Length of data                                           */
  DCL len FIXED BIN(31);
/* Receiver variable                                        */
/*   for binary numbers                                     */
  DCL charbin4 CHARACTER(4);             /* charbin4 allows a    */
  DCL int FIXED BIN(31) BASED(numptr);   /* number to be returned */
  DCL numptr POINTER;                    /* in int by setting    */
                                         /* numptr to the address */
/* NOTE: This same technique is used with charbin4 and int to */
/* retrieve binary numbers from the attribute information table */
/* of a directory entry.                                    */
/*                                                          */
/* Receiver variable                                        */
/*   for file system names                                  */
  DCL FSName CHARACTER(10);
/* The error code is the same as for the above APIs.        */
/************************************************************/


/* entrypos is the offset to the start of the list produced by */
/* QHFLSTFS.                                                */
  DCL entrypos FIXED BIN(31);

/* numentries is the number of file systems returned by QHFLSTFS. */
  DCL numentries FIXED BIN(31);

/* entrylen is the size of each entry returned by QHFLSTFS. */
  DCL entrylen FIXED BIN(31);

/* att is used to work with the information in a directory entry. */
  DCL att CHARACTER(276);

/* name is the name of the directory to list.               */
  DCL name CHARACTER(100);

/* attname and attval are the name and value for an attribute. */
/* attnamelen and attvallen are the respective lengths.     */
  DCL attname CHARACTER(30);
  DCL attval CHARACTER(30);
  DCL attnamelen FIXED BIN(31);
  DCL attvallen FIXED BIN(31);

/* newname is the value of the attribute QNAME.             */
  DCL newname CHARACTER(30);

/* filesize is the value of the attribute QFILSIZE.         */
  DCL filesize FIXED BIN(31);

/* fileatt is the value of the attribute QFILATTR.          */
  DCL fileatt CHARACTER(10);

/* chartime is the date/time value for QCRTDTTM.            */
  DCL 1 chartime,
        3 century CHARACTER(1),
        3 year    CHARACTER(2),
        3 month   CHARACTER(2),
        3 day     CHARACTER(2),
        3 hour    CHARACTER(2),
        3 minute  CHARACTER(2),
        3 second  CHARACTER(2);

/* bytes_used is used to find the beginning of attributes in a */
/* directory entry.                                         */
  DCL bytes_used FIXED BIN(31);

/* Loop control variable                                    */
  DCL i FIXED BIN(15);



  numptr = ADDR(charbin4);           /* Set numptr, Buf_ptr,  */
  Buf_ptr = ADDR(buffer);            /* SelPtr, and Error_Code. */
```

```
Sel_ptr = ADDR(selection);
Error_Code = 0;
name = SUBSTR(Dir_Name,1,INDEX(Dir_Name,' ')); /* Set name       */

/* If name is blank, list the file systems available.           */
IF name = ' ' THEN
    DO;
    name = 'ROOT';
    PUT SKIP EDIT('Directory listing for path ',name) (a(27),a(53));
    PUT SKIP;
    PUT SKIP;
    /* Create the user space for QHFLSTFS to use.               */
    size = 1;
    text = 'temporary user space used by program DIR';
    CALL QUSCRTUS('FSLST     QTEMP    ','TEMPSPACE ',size,' ',
                  '*USE      ',text,'*YES      ',Error_Code);

    /* List the file systems into that space.                   */
    CALL QHFLSTFS('FSLST     QTEMP    ','HFSL0100',Error_Code);

    /* Get the starting point for the file system entries.      */
    startpos = 125;
    len = 4;
    CALL QUSRTVUS('FSLST     QTEMP    ',startpos,len,charbin4,
                  Error_Code);
    entrypos = int;

    /* Get the number of entries in the user space.             */
    startpos = 133;
    len = 4;
    CALL QUSRTVUS('FSLST     QTEMP    ',startpos,len,charbin4,
                  Error_Code);
    numentries = int;

    /* Find the length of the entries.                          */
    startpos = 137;
    len = 4;
    CALL QUSRTVUS('FSLST     QTEMP    ',startpos,len,charbin4,
                  Error_Code);
    entrylen = int;

    /* Loop through the entries and get the names of the file   */
    /* systems.                                                 */
    DO i = 1 to numentries;
      startpos = entrypos + 1;
      len = 10;
      CALL QUSRTVUS('FSLST     QTEMP    ',startpos,len,FSname,
                    Error_Code);
      /* List the names into the spooled file.                  */
      PUT SKIP EDIT(FSname,'<DIR>') ( a(20),A(5) );
      entrypos = entrypos + entrylen;
    END;
    END;
/* If the name was not blank, list that directory.              */
ELSE
    DO;
    PUT SKIP EDIT('Directory listing for path ',name) (a(27),a(53));
    PUT SKIP;
    PUT SKIP;
    /* Build the attribute selection table for QHFOPNDR.   */
    selection_Struct.num_att   = 3;
    selection_Struct.offset1   = 16;
    selection_Struct.offset2   = 28;
    selection_Struct.offset3   = 40;
    selection_Struct.att_len1  = 8;
    selection_Struct.att_name1 = 'QFILSIZE';
    selection_Struct.att_len2  = 8;
    selection_Struct.att_name2 = 'QCRTDTTM';
    selection_Struct.att_len3  = 8;
    selection_Struct.att_name3 = 'QFILATTR';
    selectionlen = 52;
    openinfo = '10    ';                  /* Set openinfo.      */
    namelen = INDEX(NAME,' ') - 1;        /* Set namelen.       */
    /* Open the directory.                                      */
    Call QHFOPNDR(Dir_handle,name,namelen,openinfo,selection,
                  selectionlen,Error_Code);

    /* Read one entry from the directory.                       */
    Call QHFRDDR(Dir_handle,Buffer,300,1,result_count,
                 bytes_returned,Error_Code);

    DO WHILE (result_count > 0);
       attname = '                          ';
       attval  = '                          ';
       att = buffer_Struct.attinfo; /* Set att to Attinfo.      */
       bytes_used = 20; /* 20 equals the amount of data before  */
                        /* attinfo in buffer_struct.            */
       /* Loop for the number of attributes in the entry.      */
       DO i = 1 TO buffer_Struct.num_att;
         charbin4 = SUBSTR(att,1,4);   /* Get attnamelen from   */
         attnamelen = int;             /* the attribute entry.  */
         att = SUBSTR(att,5);          /* Update att.           */
         bytes_used = bytes_used + 4;  /* Update bytes_used.    */
         charbin4 = SUBSTR(att,1,4);   /* Get attvallen from    */
         attvallen = int;              /* the attribute entry.  */
         att = SUBSTR(att,9);          /* Update att.           */
         bytes_used = bytes_used + 8;  /* Update bytes_used.    */
         attname = SUBSTR(att,1,attnamelen);  /* Get attname.   */
         att = SUBSTR(att,attnamelen + 1);    /* Update att.    */
         bytes_used = bytes_used + attnamelen;/* Update bytes_used.*/
         attval = SUBSTR(att,1,attvallen);    /* Get attval.    */
         att = SUBSTR(att,attvallen + 1);     /* Update att.    */
         bytes_used = bytes_used + attvallen; /* Update bytes_used.*/
         /* Update att so that its first character is the first */
         /* character of the next attribute entry.              */
          IF (bytes_used := buffer_struct.offsets(i+1)) &
             ( i := buffer_Struct.num_att) THEN
             att = SUBSTR(att,buffer_struct.offsets(i)-bytes_used+1);

          /* If the attribute is QNAME, then set newname.     */
          IF attname = 'QNAME' THEN
             DO;
             newname = attval;
             END;

          /* If the attribute is QFILSIZE, then set filesize. */
          ELSE IF attname = 'QFILSIZE' THEN
             DO;
             charbin4 = SUBSTR(attval,1,4);
             filesize = int;
             END;

          /* If it was QCRTDTTM, then set time.               */
          ELSE IF attname = 'QCRTDTTM' THEN
             chartime = SUBSTR(attval,1,13);
          /* Else the attribute was QFILATTR, so set fileatt. */
          ELSE
             fileatt = attval;
       END; /* DO */

       /* If the entry was a directory, list its name and      */
       /* and <DIR>.                                           */
       IF (SUBSTR(fileatt,4,1) = '1') THEN
          DO;
          PUT SKIP;
          PUT EDIT(newname,'<DIR>') (a(12),x(3),a(5));
          END;

       /* If the entry is not a hidden file, list its name     */
       /* and size.                                            */
       ELSE IF (SUBSTR(fileatt,2,1) := '1') THEN
          DO;
          PUT SKIP;
          PUT EDIT(newname,filesize) (a(12),f(8));
          END;

       /* If the entry is not a hidden file or directory, list  */
       /* its date of creation.                                 */
       IF (SUBSTR(fileatt,2,1) := '1') THEN
          DO;
            PUT EDIT(month,'-',day,'-',year)
                   (x(4),a(2),a(1),a(2),a(1),a(2));
            PUT EDIT(hour,':',minute,':',second)
                   (x(3),A(2),a(1),a(2),a(1),a(2));
          END;
       Call QHFRDDR(Dir_handle,Buffer,200,1,result_count,
                    bytes_returned,Error_Code);
    END; /* while */
    END; /* ELSE DO */
/* Close the directory. */
CALL QHFCLODR(Dir_handle,Error_Code);
END DIR;
```

Listing Subdirectories

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  List the subdirectories of the path passed to the      */
/*            program to a spooled file.                             */
/*                                                                   */
/* LANGUAGE:  PL/I                                                   */
/*                                                                   */
/* APIs USED: QHFOPNDR, QHFCLODR, QHFRDDR                            */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

DSPSUBDR: PROCEDURE(Dir_Name) OPTIONS(MAIN);


/* Parameter declaration                                             */
DCL Dir_Name CHARACTER (100);

/* API entry declaration                                             */
/*                                                                   */
/* Note that the last parameter, the error code, is declared as      */
/* FIXED BIN(31) and is always zero in the calls to the APIs.        */
/* This specifies that the bytes available are zero and              */
/* exceptions should be signaled.                                    */

DCL QHFOPNDR ENTRY(CHAR(16),CHAR(100),FIXED BIN(31),CHAR(6),
                   CHAR(20),FIXED BIN(31),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

DCL QHFCLODR ENTRY(CHAR(16),FIXED BIN(31)) OPTIONS(ASSEMBLER);

DCL QHFRDDR ENTRY(CHAR(16),CHAR(200),FIXED BIN(31),FIXED BIN(31),
                  FIXED BIN(31),FIXED BIN(31),FIXED BIN(31))
                  OPTIONS(ASSEMBLER);


ON ERROR SYSTEM;

/* Take only the part of the parameter up until the first blank, */
/* then print the heading to a spooled file and call the         */
/* the procedure to print the rest of the subdirectories.        */

Dir_Name = SUBSTR(Dir_Name,1,INDEX(Dir_Name,' '));
PUT SKIP EDIT('directory substructure starting at ',Dir_Name)
             (A(35),a(45));
CALL Print_Subdir(Dir_Name,0);

/*********************************************************************/
/*  Print_SubDir                                                     */
/*********************************************************************/
/* This procedure prints out the subdirectory name and reads all */
/* directory entries in name, calling itself recursively         */
/* for any entry that is itself a subdirectory.                  */
/*********************************************************************/
/*********************************************************************/
Print_SubDir: PROCEDURE (name,numtabs);

  /* Parameter declarations                                          */
  DCL name CHARACTER (100);
  DCL numtabs FIXED BIN(15);

  /*******************************************************************/
  /* Parameters for QHFOPNDR                                         */
  /*******************************************************************/
  /* Directory handle                                                */
  DCL Dir_Handle CHAR(16);
  /* The paramter name passed to Print_SubDir is used for the      */
  /* path name.                                                      */
  /*                                                                 */
  /* Length of directory name                                        */
  DCL namelen FIXED BIN(31);
  /* Open information                                                */
  DCL openinfo CHAR(6);
  /* Attribute selection table                                       */
  DCL selection CHARACTER(20);            /* selection is used   */
  DCL 1 selection_Struct BASED(Sel_ptr),  /* to allocate storage */
         2 num_att FIXED BIN(31),         /* for selection_Struct*/
         2 offset  FIXED BIN(31),         /* this is done by     */
         2 att_len FIXED BIN(31),         /* setting Sel_ptr to  */
         2 att_name CHAR(8);              /* the address of      */
  DCL Sel_ptr POINTER;                    /* selection           */
  /* Length of attribute selection table                             */
  DCL selectionlen FIXED BIN(31);
  /* Error code                                                      */
  DCL Error_Code FIXED BIN(31);
  /*******************************************************************/


  /*******************************************************************/
  /* Parameters for QHFRDDR                                          */
  /*******************************************************************/
  /* The directory handle is the same as that for QHFOPNDR.          */
  /*                                                                 */
  /* Data buffer                                                     */
  DCL buffer CHARACTER(200);          /* buffer allocates storage */
  DCL 1 buffer_Struct BASED(Buf_ptr),  /* for buffer_Struct       */
         2 num_dir_ret FIXED BIN(31), /* by setting Buf_ptr to   */
         2 offset_dir  FIXED BIN(31), /* the address of buffer.  */
         2 num_att     FIXED BIN(31),
         2 offsets(2)  FIXED BIN(31),
         2 attinfo     CHARACTER(180);
  DCL Buf_ptr POINTER;
  /* Length of data buffer and number of entries to retrieve are     */
  /* hard coded into the API call.                                   */
  /*                                                                 */
  /* Number of directory entries retrieved                           */
  DCL result_count FIXED BIN(31);
  /* Length of data returned                                         */
  DCL bytes_returned FIXED BIN(31);
  /* The error code is the same as is used for QHFOPNDR.             */
  /*******************************************************************/

  /*******************************************************************/
  /* Parameters for QHFCLODR.                                        */
  /*******************************************************************/
  /* Both the directory handle and the error code are the same as    */
  /* those used for QHFOPNDR and QHFRDDR.                            */
  /*******************************************************************/

  /* charbin4, int, and numptr are used in conjunction to retrieve */
  /* binary nubers from the datat buffer.                          */
  /* This is done by setting numptr to the address of charbin4.    */
  /* Then, when four bytes are put into charbin4 using SUBSTR, it  */
  /* has the effect of reading a binary integer from the buffer.   */
  DCL charbin4 CHARACTER(4);
  DCL int FIXED BIN(31) BASED(numptr);
  DCL numptr POINTER;

  /* bytes_used is used to keep track of the number of bytes used */
  /* from the data buffer.  This is used to get to the             */
  /* beginning of each attribute.                                  */
  DCL bytes_used FIXED BIN(31);

  /* The name and length of the name of an attribute               */
  DCL attname CHARACTER(30);
  DCL attnamelen FIXED BIN(31);

  /* The value and length of the value of an attribute             */
  DCL attvallen FIXED BIN(31);
  DCL attval CHARACTER(30);

  /* The name and length of the name for the attribute QNAME       */
  DCL newname CHARACTER(30);
  DCL newnamelen FIXED BIN(31);

  /* The value of the attribute QFILATTR                           */
  DCL fileatt CHARACTER(9);

  /* Used to manipulate the information in the data buffer         */
  DCL att CHARACTER(180);

  /* tab is set to blanks and used to tab subdirectory entries.    */
  DCL tab CHAR(5);

  /* Loop control variables                                        */
  DCL i FIXED BIN(15);
  DCL j FIXED BIN(15);

/* Start of code for Print_SubDir                                    */
Sel_ptr = ADDR(selection);       /* Set ptr for selection table.  */
Buf_ptr = ADDR(buffer);          /* Set ptr for buffer.           */
openinfo = '10    ';             /* Set open information.         */
selection_Struct.num_att = 1;    /* Constuct the selection table. */
```

```
selection_Struct.offset  = 8;
selection_struct.att_len = 8;
selection_struct.att_name = 'QFILATTR';
selectionlen = 20;              /* Set the length of the table.  */
  tab = '     ';                /* tab = 5 spaces.               */
  numptr = ADDR(charbin4);      /* Set address for int.          */
  namelen = INDEX(name,' ') - 1; /* Set namelen.                 */
  Error_Code = 0;               /* Set error code for exceptions.*/

  /* Open the directory.                                        */
  Call QHFOPNDR(Dir_handle,name,namelen,openinfo,selection,
               selectionlen,Error_Code);

  /* Read one directory entry.                                  */
  Call QHFRDDR(Dir_handle,Buffer,200,1,result_count,
              bytes_returned,Error_Code);

  PUT SKIP;                     /* Move to next line.           */
  DO j = 1 TO NUMTABS;          /* List numtabs tabs.           */
     PUT EDIT(tab) (a(5));
  END;
  PUT EDIT(name) (a(30));       /* Write directory name         */
                                /* to splooled file.            */

  DO WHILE (result_count > 0);/* While read directory was successful*/
     attname = '                          ';
     attval  = '                          ';
     att = buffer_Struct.attinfo; /* Set att to buffer_Struct.attinfo*/
     bytes_used = 12; /* Number of bytes in att table before attinfo */
     DO i = 1 TO buffer_Struct.num_att;
        charbin4 = SUBSTR(att,1,4); /* Read attribute name length    */
        attnamelen = int;           /* from att and set attnamelen.  */
        att = SUBSTR(att,5);        /* Update att and bytes_used.    */
        bytes_used = bytes_used + 4;
        charbin4 = SUBSTR(att,1,4); /* Read attribute value length   */
        attvallen = int;            /* from att and set attvallen.   */
        att = SUBSTR(att,9);        /* Update att and bytes_used.    */
        bytes_used = bytes_used + 8;
        attname = SUBSTR(att,1,attnamelen);   /* Set attname.        */
        att = SUBSTR(att,attnamelen + 1);     /* Update att.         */
        bytes_used = bytes_used + attnamelen; /* Update bytes_used.  */
        attval = SUBSTR(att,1,attvallen);     /* Set attval.         */
        att = SUBSTR(att,attvallen + 1);      /* Update att and      */
        bytes_used = bytes_used + attvallen;  /* bytes_used.         */

     /* Set att to next attribute using bytes_used and the next      */
     /* attribute's offset.                                          */
        IF (bytes_used := buffer_struct.offsets(i+1)) &
           ( i := buffer_Struct.num_att) THEN
           att = SUBSTR(att,buffer_struct.offsets(i) - bytes_used + 1);

     /* If attribute is QNAME, set newname and newnamelen            */
     /* in case the entry is a directory.                            */
        IF attname = 'QNAME' THEN
           DO;
           newname = attval;
           newnamelen = attvallen;
           END;
     /* Else attribute was QFILATTR, so set fileatt.                 */
        ELSE
           fileatt = attval;
     END; /* DO */

     /* If the entry was a directory                                 */
     IF (SUBSTR(fileatt,4,1) = '1') THEN
        DO;
        /* Construct path name and call Print_SubDir to print        */
        /* its subdirectories.                                       */
        newname = SUBSTR(name,1,namelen)||'/'||
                  SUBSTR(newname,1,newnamelen);
        CALL Print_Subdir(newname,numtabs + 1);
        END;
     /* Read next directory entry                                    */
     Call QHFRDDR(Dir_handle,Buffer,200,1,result_count,
                 bytes_returned,Error_Code);
  END; /* while */
  /* Close the directory                                           */
  CALL QHFCLODR(Dir_Handle,Error_Code);
END Print_Subdir;


END DSPSUBDR;
```

Working with Stream Files

The following C/400 program performs the following functions:

- Opens an existing stream file
- Creates or replaces a database file
- Reads from the stream file and writes to the database file until end-of-file
- Closes both files

The program uses the following hierarchical file system (HFS) APIs:

- Open Stream File (QHFOPNSF)
- Read from Stream File (QHFRDSF)
- Close Stream File (QHFCLOSF)

```
/*****************************************************************/
/* Program Name:  HFSCOPY  C                                     */
/* Language    :  C/400                                          */
/* Description :  This program will do the following:            */
/*                -- Create or replace a stream file             */
/*                -- Create or replace a data base file          */
/*                -- Read from the stream file and write to the  */
/*                   data base file until EOF                    */
/*                -- Close both files when done                  */
/*****************************************************************/

/*****************************************************************/
/* Include files                                                 */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ophfapi.h>


/*****************************************************************/
/* Structure and variable definitions                            */
/*****************************************************************/

#define ON   1
#define OFF  0
typedef struct error_code_struct {
               int  bytes_provided;
               int  bytes_available;
               char exception_id[7];
               char reserved_field;
               char exception_data[256];
              }error_code_struct;
error_code_struct error_code;
handle_struct file_handle;
char path_name[30];
char open_info[10];
char attrib_info;
char action;
char read_buffer[80];
int  path_length;
int  attrib_length = 0;
int  bytes_to_read;
int  bytes_read = 0;
int end_file;
int cmpgood;
FILE *FP;

/*****************************************************************/
/*printErrCode: Routine to print the error code structure        */
/*****************************************************************/
void printErrCode(theErrCode)
error_code_struct *theErrCode;
{
   int i;
   char *tempptr = theErrCode->exception_id;
   printf("Bytes Provided -> %d\n",theErrCode->bytes_provided);
   printf("Bytes Available -> %d\n",theErrCode->bytes_available);
   printf("Exception ID -> ");
   for (i=0;i<7 ;i++,tempptr++ )
     {
```

```
      putchar(*tempptr);
    }
  putchar('\n');
}


/*********************************************************************/
/* Start of code                                                     */
/*********************************************************************/
main()
{

 error_code.bytes_provided = 116;
 strcpy(path_name,"/QDLS/HFSFLR/SAMPLE.HFS");
 path_length = strlen(path_name);

/*********************************************************************/
/* Open the stream file                                              */
/*********************************************************************/
 strcpy(open_info,"210 120   ");    /* Create or replace the file   */
 printf("OPEN STREAM FILE:\n  ");
 QHFOPNSF(&file_handle,
          path_name,
          path_length,
          open_info,
          &attrib_info,
          attrib_length,
          &action,
          &error_code);
 if (error_code.bytes_available := 0)
   {
    printErrCode(&error_code);
    exit(1);
   }

/*********************************************************************/
/* Open a data base file                                             */
/*********************************************************************/

 if (( FP = fopen("HFSLIB/HFSFILE(SAMPLE)","wb")) == NULL)
   {
    printf("Cannot open HFSLIB/HFSFILE(SAMPLE)\n");
    exit(1);
   }

/*********************************************************************/
/* Loop through reading from the stream file and writing to the      */
/* data base file.                                                   */
/*********************************************************************/

 end_file = OFF;
 while (end_file == OFF)
   {
     /*************************************************************/
     /* Read 80 bytes from the stream file                        */
     /*************************************************************/
     bytes_to_read = 80;
     printf("READ STREAM FILE:\n  ");
     QHFRDSF(&file_handle,
             read_buffer,
             bytes_to_read,
             &bytes_read,
             &error_code);
     if (error_code.bytes_available := 0)
       {
         cmpgood = strncmp("CPF1F33",error_code.exception_id,7);
         if (cmpgood := 0)
           printErrCode(&error_code);
         end_file = ON;
       }
     else
       {
         printf("BYTES READ: %d\n  ",bytes_read);
         printf("READ BUFFER: %s\n",read_buffer);
         if (bytes_read < bytes_to_read)
           {
             end_file = ON;
             /*****************************************************/
             /* Write remaining bytes to the data base file       */
             /*****************************************************/
             if (bytes_read > 0)
               fwrite(read_buffer,1,bytes_read,FP);
           }
       }
   }

/*********************************************************************/
/* Close the stream file                                             */
/*********************************************************************/
 printf("CLOSE STREAM FILE:\n  ");
 QHFCLOSF(&file_handle,
          &error_code);
 if (error_code.bytes_available := 0)
   printErrCode(&error_code);

/*********************************************************************/
/* Close the data base file                                          */
/*********************************************************************/
 fclose(FP);
}
```

Using SNA Management Services Transport APIs

This example shows a source and target application using network management transport APIs to send and receive management services data.

Source Application Program

This source application program sends a request to a target application.

```
/*-------------------------------------------------------------------*/
/* This is a source application that uses the management services    */
/* transport APIs.  It does the following:                           */
/*   1.  Prompts for the network ID and CP name of the remote system */
/*       where target application MSTTARG has been started.          */
/*   2.  Prompts for data to be sent to MSTTARG.                     */
/*   3.  Prompts for whether or not a reply is required.             */
/*   4.  Sends a management services transport request to MSTTARG.   */
/*   5.  Repeats steps 2-4 until QUIT is entered.                    */
/*                                                                   */
/* Note: MSTTARG may be ended by this application by sending it the  */
/*       the string "ENDRMTAPP".                                     */
/*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------*/
/* Includes                                                          */
/*-------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define NOERROR "NOERROR"
#define RQSONLY "*RQS      "
#define RQSRPY "*RQSRPY   "

/*-------------------------------------------------------------------*/
/* Type definitions                                                  */
/*-------------------------------------------------------------------*/
typedef int HANDLE;                 /* typedef for handle            */
typedef char APPLNAME[8];           /* typedef for application name  */
typedef char NETID[8];              /* typedef for network ID        */
typedef char CPNAME[8];             /* typedef for control point name*/
typedef char MODENAME[8];           /* typedef for mode name         */
typedef char SENSECODE[8];          /* typedef for SNA sense code (in
                                       character format)            */
typedef char LIBNAME[10];           /* typedef for library name      */
typedef char QNAME[10];             /* typedef for data queue name   */
typedef char MSGID[7];              /* typedef for message ID        */
typedef char EXCPDATA[48];          /* typedef for exception data    */
typedef char CATEGORY[8];           /* typedef for category          */
typedef char APPLTYPE[10];          /* typedef for application type  */
typedef char REPLREG[10];           /* typedef for replace
                                       registration                 */
typedef char DATARCVD[10];          /* typedef for data received     */
typedef char REQTYPE[10];           /* typedef for request type      */
typedef char POSTRPL[10];           /* typedef for post reply        */
typedef char REQUESTID[53];         /* typedef for request ID        */
typedef char SRBUFFER[500];         /* typedef for send/receive
                                       buffer.  This program limits
                                       the amount of data to be sent
```

```
                                or received to 500 bytes.  The
                                maximum size of a management
                                services transport buffer is
                                31739.                          */

typedef struct {                /* Library-qualified data queue
                                   name                         */
   QNAME data_queue_name;       /* data queue name              */
   LIBNAME library_name;        /* library name                 */
  } QUALQNAME;

typedef struct {                /* Error code structure         */
   int bytes_provided;          /* number of bytes provided     */
   int bytes_available;         /* number of bytes available    */
   MSGID exception_ID;          /* exception ID                 */
   char reserved_area;          /* reserved                     */
   EXCPDATA exception_data;     /* exception data               */
  } ERRORCODE;

typedef struct {                /* Notification record structure */
   char record_type[10];        /* Record type                  */
   char function[2];            /* Function                     */
   HANDLE handle;               /* Handle                       */
   REQUESTID req_id;            /* Request ID                   */
   char reserved[11];           /* Reserved area                */
  } NOTIFRCD;

typedef struct {                /* Receiver variable structure  */
   int bytes_provided;          /* number of bytes provided     */
   int bytes_available;         /* number of bytes available    */
   SRBUFFER received_data;      /* received data                */
  } RECEIVERVAR;

typedef struct {                /* Qualified application name   */
   NETID network_id;            /* Network ID                   */
   CPNAME cp_name;              /* Control point name           */
   APPLNAME app_name;           /* Application name             */
  } QUALAPPL;

/*--------------------------------------------------------------------*/
/* External program declarations                                      */
/*--------------------------------------------------------------------*/
#pragma linkage(QNMSTRAP, OS)       /* Start application API      */
extern void QNMSTRAP (HANDLE *handle, /* pointer to handle        */
                      APPLNAME *applname, /* pointer to appl name */
                      QUALQNAME *qualqname, /* pointer to data queue
                                   name                         */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                    */

#pragma linkage(QNMENDAP, OS)       /* End application API        */
extern void QNMENDAP (HANDLE *handle, /* pointer to handle        */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                    */

#pragma linkage(QNMRCVDT, OS)       /* Receive data API           */
extern void QNMRCVDT (HANDLE *handle, /* pointer to handle        */
                      RECEIVERVAR *rcvvar, /* pointer to receiver
                                   variable                     */
                      int *rcvvarln, /* pointer to receiver variable
                                   length                       */
                      REQUESTID *reqid, /* pointer to request ID */
                      QUALAPPL *qualappl, /* pointer to remote
                                   application name             */
                      DATARCVD *datarcvd, /* pointer to type of data
                                   received                     */
                      int *waittim, /* pointer to wait time       */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                    */

#pragma linkage(QNMSNDRQ, OS)       /* Send request API           */
extern void QNMSNDRQ (HANDLE *handle, /* pointer to handle        */
                      QUALAPPL *qualappl, /* pointer to remote
                                   application name             */
                      REQUESTID *reqid, /* pointer to request ID */
                      SRBUFFER *sndbuf, /* pointer to send buffer */
                      int *sndbufln, /* pointer to send buffer length*/
                      REQTYPE *reqtype, /* pointer to request type */
                      POSTRPL *postrpl, /* pointer to post reply  */
                      int *waittim, /* pointer to wait time       */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                    */

#pragma linkage(QNMCHGMN, OS)       /* Change mode name API       */
extern void QNMCHGMN (HANDLE *handle, /* pointer to handle        */
                      MODENAME *modename, /* pointer to mode name */
                      ERRORCODE *errorcode); /* pointer to error code
                                   parameter                    */

void check_error_code (char func_name[8]); /* Used to check error code
                                                                  */
void get_network_id (void);         /* Get network ID of destination
                                   node                         */
void get_cp_name (void);            /* Get CP name of destination
                                   node                         */
void process_replies(void);         /* Process replies received from
                                   destination application      */

/*--------------------------------------------------------------------*/
/* Global declarations                                                */
/*--------------------------------------------------------------------*/
 HANDLE appl_handle;                /* Handle of application      */
 ERRORCODE error_code_struc =       /* Error code parameter       */
          {sizeof(error_code_struc), /* Initialize bytes provided */
           0,                       /* initialize bytes available */
           NOERROR};                /* initialize error code      */
 char input_line[80];               /* Input data                 */
 QUALAPPL qual_appl =               /* Qualified application name */
       {"        ","        ","        "};
 REQUESTID req_id;                  /* Returned request ID        */
 int wait_time = -1;                /* Wait time = wait forever   */

/*--------------------------------------------------------------------*/
/* Start of main.                                                     */
/*--------------------------------------------------------------------*/
int main ()
{
 APPLNAME appl_name = "MSTSOURC";   /* Application name to be used */
 QUALQNAME data_queue_parm =        /* Data queue name to be used  */
          {"*NONE     ", "         "}; /* Initialize structure    */
 NOTIFRCD notif_record;             /* Area to contain notification
                                   record                       */
 CATEGORY category = "*NONE   ";    /* SNA/Management Services function
                                   set group                    */
 APPLTYPE appl_type = "*FPAPP    "; /* Application type           */
 REPLREG replace_reg = "*YES      "; /* Replace registration = *YES */
 int sys_result;                    /* Result of system function  */
 char end_msg[] = "ENDRMTAPPL";     /* If this data is received then
                                   the application will end     */
 char incoming_data[] = "01";       /* Incoming data constant     */
 SRBUFFER send_buffer;              /* Send buffer                */
 int data_length;                   /* Length of send data        */
 char input_char;                   /* Input character            */
 REQTYPE req_type;                  /* Request type               */
 POSTRPL post_reply = "*NO       "; /* Don't post any received replies
                                                                  */
 MODENAME mode_name = "#INTER  ";   /* Mode name = #INTER         */

/*--------------------------------------------------------------------*/
/* Start of code                                                      */
/*--------------------------------------------------------------------*/
 QNMSTRAP (&appl_handle,
           &appl_name,
           &data_queue_parm,
           &error_code_struc);      /* Start application          */
 check_error_code("QNMSTRAP");      /* Check error code           */
 QNMCHGMN (&appl_handle,
           &mode_name,
           &error_code_struc);      /* Change mode name           */
 check_error_code("QNMCHGMN");      /* Check error code           */
 get_network_id();                  /* Get network ID             */
 get_cp_name();                     /* Get CP name                */
 memcpy(qual_appl.app_name,
        "MSTTARG ",
        sizeof(qual_appl.app_name)); /* Copy application name     */
 printf ("Enter message to send to remote application or "
         "QUIT to end\n");
 gets(input_line);
 while (memcmp(input_line,
               "QUIT",
               sizeof("QUIT")) != 0) /* While an ending string
                                   has not been entered         */
   {
     data_length = strlen(input_line); /* Get length of message */
     memcpy(send_buffer,
            input_line,
```

```
          data_length);           /* Put message in send buffer    */
    printf("Reply necessary? (Y or N)\n"); /* Prompt for reply
                                      indicator                     */
    gets(input_line);             /* Get reply character           */
    input_char = toupper(input_line[0]); /* Convert character to
                                      uppercase                     */
    while (strlen(input_line) != 1 ||
           (input_char != 'Y' &&
           input_char != 'N'))
      {
        printf("Please type Y or N\n");
        gets(input_line);         /* Get reply character           */
        input_char = toupper(input_line[0]); /* Convert character to
                                      uppercase                     */
      }
    if (input_char == 'Y')
      {
        memcpy(req_type,
               RQSRPY,
               sizeof(req_type)); /* Indicate request should have
                                      a reply                       */
      }
    else
      {
        memcpy(req_type,
               RQSONLY,
               sizeof(req_type)); /* Indicate request should not have
                                      a reply                       */
      }
    QNMSNDRQ (&appl_handle,
              &qual_appl,
              &req_id,
              &send_buffer,
              &data_length,
              &req_type,
              &post_reply,
              &wait_time,
              &error_code_struc); /* Send request to remote
                                      application                   */
    check_error_code("QNMSNDRQ"); /* Check error code              */
    if (input_char == 'Y')
      {
        process_replies();        /* Process one or more received
                                      replies                       */
      }
    printf ("Enter message to send to remote application or "
            "QUIT to end\n");
    gets(input_line);
  }
 QNMENDAP (&appl_handle,
           &error_code_struc);    /* End the application           */

 return 0;
}

/*--------------------------------------------------------------------*/
/* process_replies function                                           */
/*--------------------------------------------------------------------*/
void process_replies ()
{
 RECEIVERVAR receiver_var =       /* Receiver variable             */
          {sizeof(receiver_var)}; /* Initialize bytes provided     */
 int rcv_var_len = sizeof(receiver_var); /* Length of receiver
                                      variable                      */
 DATARCVD data_rcvd = "*NODATA   "; /* Type of data received        */
 QUALAPPL qual_appl;              /* Sender of reply               */

 printf ("Received reply(s):\n");
 while (memcmp(data_rcvd,
               "*RPYCPL   ",
               sizeof(data_rcvd)) != 0) /* While final reply has not
                                      been received                 */
   {
     strncpy(receiver_var.received_data,
             "\0",
             sizeof(receiver_var.received_data)); /* Null out
                                      data buffer                   */
     QNMRCVDT (&appl_handle,
               &receiver_var,
               &rcv_var_len,
               &req_id,
               &qual_appl,
               &data_rcvd,
               &wait_time,
               &error_code_struc); /* Receive reply                */
     check_error_code("QNMRCVDT"); /* Check error code             */
     printf("%1.500s\n",receiver_var.received_data); /* Print out
                                                        reply       */
   }
}

/*--------------------------------------------------------------------*/
/* get_network_id function.                                           */
/*--------------------------------------------------------------------*/
void get_network_id ()
{
 int count;
 printf("Enter network ID of remote system where MSTTARG "
        "application has been started\n"); /* Prompt for network
                                      ID                            */
 gets(input_line);                /* Get network ID                */
 while (strlen(input_line) <= 0 ||
        strlen(input_line) > 8)   /* While network ID is not valid */
   {
     printf("Network ID is too long or too short - try again\n");
     gets(input_line);            /* Get network ID                */
   }
 memcpy(qual_appl.network_id,
        input_line,
        strlen(input_line));      /* Copy network ID               */
 for (count=0; count < strlen(input_line); count++)
   qual_appl.network_id[count] =
                toupper(qual_appl.network_id[count]); /* Convert
                                      input to uppercase            */
}

/*--------------------------------------------------------------------*/
/* get_cp_name function.                                              */
/*--------------------------------------------------------------------*/
void get_cp_name ()
{
 int count;
 printf("Enter CP name of remote system where MSTTARG application "
        "has been started\n");    /* Prompt for CP name            */
 gets(input_line);                /* Get CP name                   */
 while (strlen(input_line) <= 0 ||
        strlen(input_line) > 8)   /* While CP name is not valid    */
   {
     printf("CP name is too long or too short - try again\n");
     gets(input_line);            /* Get CP name                   */
   }
 memcpy(qual_appl.cp_name,
        input_line,
        strlen(input_line));      /* Copy CP name                  */
 for (count=0; count < strlen(input_line); count++)
   qual_appl.cp_name[count] =
                toupper(qual_appl.cp_name[count]); /* Convert
                                      input to uppercase            */
}

/*--------------------------------------------------------------------*/
/* check_error_code -                                                 */
/*--------------------------------------------------------------------*/
void check_error_code (char func_name[8])
{
 char *sense_ptr = error_code_struc.exception_data + 36; /*
                                      Pointer to sense code in
                                      exception data                */
 SENSECODE sense_code;            /* SNA sense code                */
 if (error_code_struc.bytes_available != 0) /* Error occurred?     */
   {
     printf("\n\nError occurred calling %1.8s.\n",func_name);
     memcpy(sense_code,
            sense_ptr,
            sizeof(sense_code));  /* Copy sense code from exception
                                      data                          */
     printf("Error code is %1.7s, SNA sense code is %1.8s.\n",
                 error_code_struc.exception_ID,
                 sense_code);
     if (memcmp(func_name,
            "QNMSTRAP",
            sizeof(func_name)) != 0)  /* Error did not occur on
                                      start application?            */
       {
         QNMENDAP (&appl_handle,
                   &error_code_struc); /* End the application      */
```

```
      }
    exit(EXIT_FAILURE);          /* Exit this program          */
  }
}
```

Target Application Program

This target application receives requests from and returns replies to source applications.

```
/*--------------------------------------------------------------------*/
/* This is a target application that uses the management services     */
/* transport APIs.  It receives management services transport         */
/* requests from source application MSTSOURC and displays the data    */
/* contained in the request.  If the request specifies that a         */
/* reply needs to be sent, this program accepts input from the        */
/* keyboard and sends one or more replies to the source application.  */
/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
/* Includes                                                           */
/*--------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define NOERROR "NOERROR"
#define REQUEST "*RQS      "
#define REQREPLY "*RQSRPY   "
#define REPLYINC "*RPYINCPL "
#define REPLYCMP "*RPYCPL   "

/*--------------------------------------------------------------------*/
/* Type definitions                                                   */
/*--------------------------------------------------------------------*/
typedef int HANDLE;                 /* typedef for handle          */
typedef char APPLNAME[8];           /* typedef for application name  */
typedef char NETID[8];              /* typedef for network ID        */
typedef char CPNAME[8];             /* typedef for control point name*/
typedef char SENSECODE[8];          /* typedef for SNA sense code
                                       (in character format)       */
typedef char LIBNAME[10];           /* typedef for library name      */
typedef char QNAME[10];             /* typedef for data queue name   */
typedef char MSGID[7];              /* typedef for message ID        */
typedef char EXCPDATA[48];          /* typedef for exception data    */
typedef char CATEGORY[8];           /* typedef for category          */
typedef char APPLTYPE[10];          /* typedef for application type  */
typedef char REPLREG[10];           /* typedef for replace
                                       registration                */
typedef char DATARCVD[10];          /* typedef for data received     */
typedef char REPLYTYPE[10];         /* typedef for reply type        */
typedef char REQUESTID[53];         /* typedef for request ID        */
typedef char PACKED5[3];            /* typedef for PACKED(5,0) field */
typedef char SRBUFFER[500];         /* typedef for send/receive
                                       buffer.  This program limits
                                       the amount of data to be sent
                                       or received to 500 bytes.  The
                                       maximum size of a management
                                       services transport buffer is
                                       31739.                      */

typedef struct {                    /* Library-qualified data queue
                                       name                        */
   QNAME data_queue_name;           /* data queue name             */
   LIBNAME library_name;            /* library name                */
  } QUALQNAME;

typedef struct {                    /* Error code structure        */
   int bytes_provided;              /* number of bytes provided    */
   int bytes_available;             /* number of bytes available   */
   MSGID exception_ID;              /* exception ID                */
   char reserved_area;              /* reserved                    */
   EXCPDATA exception_data;         /* exception data              */
  } ERRORCODE;

typedef struct {                    /* Notification record structure */
   char record_type[10];            /* Record type                 */
   char function[2];                /* Function                    */
   HANDLE handle;                   /* Handle                      */
   REQUESTID req_id;                /* Request ID                  */
   char reserved[11];               /* Reserved area               */
  } NOTIFRCD;

typedef struct {                    /* Receiver variable structure   */
   int bytes_provided;              /* number of bytes provided      */
   int bytes_available;             /* number of bytes available     */
   SRBUFFER received_data;          /* received data                 */
  } RECEIVERVAR;

typedef struct {                    /* Qualified application name    */
   NETID network_id;                /* Network ID                    */
   CPNAME cp_name;                  /* Control point name            */
   APPLNAME app_name;               /* Application name              */
  } QUALAPPL;

/*--------------------------------------------------------------------*/
/* External program declarations                                      */
/*--------------------------------------------------------------------*/
#pragma linkage(QNMSTRAP, OS)       /* Start application API         */
extern void QNMSTRAP (HANDLE *handle, /* pointer to handle           */
                      APPLNAME *applname,  /* pointer to application
                                              name                  */
                      QUALQNAME *qualqname,/* pointer to data queue
                                              name                  */
                      ERRORCODE *errorcode); /* pointer to error code
                                              parameter             */

#pragma linkage(QNMENDAP, OS)       /* End application API           */
extern void QNMENDAP (HANDLE *handle, /* pointer to handle           */
                      ERRORCODE *errorcode); /* pointer to error code
                                              parameter             */

#pragma linkage(QNMREGAP, OS)       /* Register application API      */
extern void QNMREGAP (HANDLE *handle, /* pointer to handle           */
                      CATEGORY *category, /* pointer to category     */
                      APPLTYPE *appltype, /* pointer to application
                                              type                  */
                      REPLREG *replreg, /* pointer to replace
                                              registration parameter */
                      ERRORCODE *errorcode); /* pointer to error code
                                              parameter             */

#pragma linkage(QNMDRGAP, OS)       /* Deregister application API    */
extern void QNMDRGAP (HANDLE *handle, /* pointer to handle           */
                      ERRORCODE *errorcode); /* pointer to error code
                                               set group            */

#pragma linkage(QNMRCVDT, OS)       /* Receive data API              */
extern void QNMRCVDT (HANDLE *handle, /* pointer to handle           */
                      RECEIVERVAR *rcvvar, /* pointer to receiver
                                              variable              */
                      int *rcvvarln, /* pointer to receiver variable
                                              length                */
                      REQUESTID *reqid, /* pointer to request ID     */
                      QUALAPPL *qualappl, /* pointer to remote
                                              application name      */
                      DATARCVD *datarcvd, /* pointer to type of data
                                              received              */
                      int *waittim, /* pointer to wait time          */
                      ERRORCODE *errorcode); /* pointer to error code
                                              parameter             */

#pragma linkage(QNMSNDRP, OS)       /* Send reply API                */
extern void QNMSNDRP (HANDLE *handle, /* pointer to handle           */
                      REQUESTID *reqid, /* pointer to request ID     */
                      SRBUFFER *sndbuf, /* pointer to send buffer    */
                      int *sndbufln, /* pointer to send buffer length*/
                      REPLYTYPE *rpltype, /* pointer to reply type   */
                      int *waittim, /* pointer to wait time          */
                      ERRORCODE *errorcode); /* pointer to error code
                                              parameter             */

#pragma linkage(QNMRCVOC, OS)       /* Receive operation completion API
                                                                     */
extern void QNMRCVOC (HANDLE *handle, /* pointer to handle           */
                      REQUESTID *reqid, /* pointer to request ID     */
                      QUALAPPL *qualappl, /* pointer to remote
                                              application name      */
                      ERRORCODE *errorcode); /* pointer to error code
                                              parameter             */

#pragma linkage(QRCVDTAQ, OS)       /* Receive data queue            */
extern void QRCVDTAQ (QNAME *queue_name, /* pointer to queue name    */
                      LIBNAME *lib_name, /* pointer to library name  */
                      PACKED5 *rcd_len, /* pointer to record length  */
```

```
                    NOTIFRCD *notifrcd, /* pointer to notification
                                        record                      */
                    PACKED5 *waittime); /* pointer to wait time    */

void check_error_code (char func_name[8]); /* Used to check error
                                        code                        */

/*-------------------------------------------------------------------*/
/* Global declarations                                               */
/*-------------------------------------------------------------------*/
 HANDLE appl_handle;                /* Handle of application       */
 ERRORCODE error_code_struc =       /* Error code parameter        */
          {sizeof(error_code_struc), /* Initialize bytes provided  */
           0,                       /* initialize bytes available  */
           NOERROR};                /* initialize error code       */
/*-------------------------------------------------------------------*/
/* Start of main function                                            */
/*-------------------------------------------------------------------*/
int main ()
{
/*-------------------------------------------------------------------*/
/* Local declarations                                                */
/*-------------------------------------------------------------------*/
 APPLNAME appl_name = "MSTTARG ";   /* Application name to be used  */
 QUALQNAME data_queue_parm =        /* Data queue name to be used   */
          {"MSTDTAQ   ", "QTEMP     "}; /* Initialize structure    */
 NOTIFRCD notif_record;             /* Area to contain notification
                                       record                       */
 RECEIVERVAR receiver_var =         /* Receiver variable            */
          {sizeof(receiver_var)};   /* Initialize bytes provided    */
 QUALAPPL qual_appl;                /* Qualified application name   */
 DATARCVD data_rcvd;                /* Type of data received        */
 CATEGORY category = "*NONE   ";    /* SNA/Management Services function
                                       set group                    */
 APPLTYPE appl_type = "*FPAPP    "; /* Application type             */
 REPLREG replace_reg = "*YES      "; /* Replace registration = *NO  */
 REPLYTYPE reply_cmp = REPLYCMP;    /* Complete reply               */
 REPLYTYPE reply_inc = REPLYINC;    /* Incomplete reply             */
 int sys_result;                    /* Result of system function    */
 int rcv_var_len = sizeof(receiver_var); /* Length of receiver
                                       variable                     */
 PACKED5 wait_time_p = "\x00\x00\x1D"; /* Packed value for wait time
                                       = -1, i.e. wait forever      */
 PACKED5 record_len;                /* Length of received data queue
                                       record                       */
 int wait_forever = -1;             /* Integer value for wait time =
                                       -1, that is, wait forever    */
 int no_wait = 0;                   /* Do not wait for I/O to
                                       complete                     */
 char end_msg[] = "ENDRMTAPPL";     /* If this data is received then
                                       the application will end     */
 char incoming_data[] = "01";       /* Incoming data constant       */
 char inbuf[85];                    /* Input buffer                 */
 SRBUFFER send_buffer;              /* Send buffer for sending
                                       replies                      */
 int reply_len;                     /* Length of reply data         */

/*-------------------------------------------------------------------*/
/* Start of executable code                                          */
/*-------------------------------------------------------------------*/
 sys_result = system("DLTDTAQ DTAQ(QTEMP/MSTDTAQ)"); /* Delete
                                       previous data queue (if any) */
 sys_result = system("CRTDTAQ DTAQ(QTEMP/MSTDTAQ) MAXLEN(80)"); /*
                                       Create data queue            */
 QNMSTRAP (&appl_handle,
          &appl_name,
          &data_queue_parm,
          &error_code_struc);       /* Start application            */
 check_error_code("QNMSTRAP");      /* Check error code             */
 QNMREGAP (&appl_handle,
          &category,
          &appl_type,
          &replace_reg,
          &error_code_struc);       /* Register the application     */
 check_error_code("QNMREGAP");      /* Check error code             */
 while (memcmp(receiver_var.received_data,
               end_msg,
               sizeof(end_msg)) != 0)
   {                                /* Loop until an ending string
                                       has been sent by the requesting
                                       application                  */
     QRCVDTAQ (&data_queue_parm.data_queue_name,
               &data_queue_parm.library_name,
               &record_len,
               &notif_record,
               &wait_time_p);       /* Receive indication from data
                                       queue                        */
     if (memcmp(notif_record.function,
                incoming_data,
                sizeof(incoming_data)) == 0) /* Incoming data was
                                       received?                    */
       {
         strncpy(receiver_var.received_data,
                 "\0",
                 sizeof(receiver_var.received_data)); /* Null out the
                                       receive buffer               */
         QNMRCVDT (&appl_handle,
                   &receiver_var,
                   &rcv_var_len,
                   &notif_record.req_id,
                   &qual_appl,
                   &data_rcvd,
                   &wait_forever,
                   &error_code_struc); /* Receive data using the
                                       request ID in the notification*/
         check_error_code("QNMRCVDT"); /* Check error code           */
         printf("%1.500s\n",receiver_var.received_data); /* Display
                                       the received data            */
         if (memcmp(data_rcvd,
                    REQREPLY,
                    sizeof(data_rcvd)) == 0) /* Request requires
                                       a reply?                     */
           {
             printf("Please enter your replies (a null line "
                    "indicates that you are finished)\n"); /* Display
                                       a prompt message             */
             gets(inbuf);           /* Get the reply data           */
             reply_len = strlen(inbuf); /* Get length of reply      */
             while (reply_len != 0)  /* While no null string was input
                                                                     */
               {
                 memcpy(send_buffer,inbuf,strlen(inbuf)); /* Copy
                                       data to send buffer          */
                 QNMSNDRP (&appl_handle,
                           &notif_record.req_id,
                           &send_buffer,
                           &reply_len,
                           &reply_inc,
                           &no_wait,
                           &error_code_struc); /* Send a reply to the
                                       source application (specify
                                       "not last" reply).  The results
                                       of this operation will be
                                       obtained later using the
                                       receive operation completion
                                       API.                         */
                 gets(inbuf);       /* Get the next reply           */
                 reply_len = strlen(inbuf); /* Get length of reply  */
               }
             QNMSNDRP (&appl_handle,
                       &notif_record.req_id,
                       &send_buffer,
                       &reply_len,
                       &reply_cmp,
                       &no_wait,
                       &error_code_struc); /* Send final reply (this
                                       contains no data).  The results
                                       of this operation will be
                                       obtained later using the
                                       receive operation completion
                                       API.                         */
           }
         else
           {                        /* A reply is not required      */
             if (memcmp(data_rcvd,
                 REQUEST,
                 sizeof(data_rcvd)) != 0) /* Something other than a
                                       request was received?        */
               {
                 printf("Incorrect data was received, "
                        "data_rcvd = %1.10s\n", data_rcvd); /* Print
                                       value of data_rcvd           */
               }
           }
       }
```

```
    else
      {                          /* A send completion was received
                                    for a previous send reply
                                    operation                    */
        QNMRCVOC (&appl_handle,
                  &notif_record.req_id,
                  &qual_appl,
                  &error_code_struc);/* Receive operation completion*/
        check_error_code("QNMRCVOC"); /* Check error code         */
        printf("Reply was sent successfully.\n"); /* Error code was
                                    OK                           */
      }
   }
 QNMDRGAP (&appl_handle,
           &error_code_struc);     /* Deregister the application   */
 QNMENDAP (&appl_handle,
           &error_code_struc);     /* End the application          */

 return 0;
}

/*--------------------------------------------------------------------*/
/* check_error_code -                                                 */
/*                                                                    */
/*   This function validates the error code parameter returned on     */
/*   the call to a management services transport API program.  If     */
/*   an error occurred, it displays the error that occurred and       */
/*   ends this program.                                               */
/*--------------------------------------------------------------------*/
void check_error_code (char func_name[8])
{
 char *sense_ptr = error_code_struc.exception_data + 36; /*
                                    Pointer to sense code in
                                    exception data               */
 SENSECODE sense_code;             /* SNA sense code               */
 if (error_code_struc.bytes_available != 0) /* Error occurred?     */
   {
     printf("\nError occurred calling %1.8s.\n",func_name);
     memcpy(sense_code,
            sense_ptr,
            sizeof(sense_code));   /* Copy sense code from exception
                                    data                         */
     printf("Error code is %1.7s, SNA sense code is %1.8s.\n",
                 error_code_struc.exception_ID,
                 sense_code);
     if (memcmp(func_name,
            "QNMSTRAP",
            sizeof("QNMSTRAP")) != 0)  /* Error did not occur on
                                    start application?           */
       {
         QNMENDAP (&appl_handle,
                   &error_code_struc); /* End the application     */

       }
     exit(EXIT_FAILURE);           /* Exit this program            */
   }
}
```

Using COBOL Program to Call APIs

This example COBOL program uses the example error handler in "Error Handler for Example COBOL Program" on page A-33.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ACF24.
*****************************************************************
*                                                               *
* THE PURPOSE OF THIS PROGRAM IS TO SHOW HOW TO CALL THE        *
* VARIOUS APIs, WHILE TESTING THAT THEY WORK PROPERLY.          *
*                                                               *
*****************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-AS400.
 OBJECT-COMPUTER.  IBM-AS400.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 old.
    05  oldname       PIC X(10).
    05  oldlibr       PIC X(10).
 77 scope             PIC X VALUE "P".
 01 errparm.
    05 input-l         PIC S9(6) BINARY VALUE ZERO.
    05 output-l        PIC S9(6) BINARY VALUE ZERO.
    05 exception-id    PIC X(7).
    05 reserved        PIC X(1).
    05 exception-data PIC X(50).
 01  new.
    05  newname       PIC X(10) VALUE "ACERRF24".
    05  newlibr       PIC X(10) VALUE "UTCBL".
 77  newlib           PIC X(10).
 PROCEDURE DIVISION.
 main-proc.
     DISPLAY "in ACF24".
     PERFORM variation-01 THRU end-variation.
     STOP RUN.
 variation-01.
*****************************************************************
*                                                               *
* This variation addresses the situation where there is no      *
* pending COBOL main, so no pending error handler can exist.    *
*                                                               *
*****************************************************************
     DISPLAY "no pending so expect nothing but error LBE7052".
     MOVE SPACES TO old exception-id.
*****************************************************************
* By setting error parm > 8, expect escape message              *
* LBE7052 to be returned in error parameter.                    *
*****************************************************************
     MOVE LENGTH OF errparm TO input-l.
     CALL "QLRRTVCE" USING old scope errparm.
     IF exception-id IS NOT =  "LBE7052" THEN
       DISPLAY "** error - expected LBE7052"
     ELSE
       DISPLAY "LBE7052 was found"
     END-IF.
*****************************************************************
* Reset input-l to ZERO,  thus any further errors will cause    *
* COBOL program to stop.                                        *
*****************************************************************
     MOVE 0 TO input-l.
     MOVE SPACES TO old exception-id.
 variation-02.
*****************************************************************
*                                                               *
* This variation creates a pending run unit.  It then makes     *
* sure that no pending error handler has been set.              *
*                                                               *
*****************************************************************
     DISPLAY "create pending run unit".
     CALL "QLRCHGCM" USING errparm.
*****************************************************************
*                                                               *
* No pending error handler exists so *NONE should be            *
* returned.                                                     *
*                                                               *
*****************************************************************
```

```
     CALL "QLRRTVCE" USING old scope errparm.
     DISPLAY "Retrieved Error Handler is=" old.
     IF oldname IS NOT = "*NONE" THEN
       DISPLAY "** error - expected *NONE for error handler"
     END-IF.
     MOVE 0 TO input-l.
     MOVE SPACES TO old exception-id.
 variation-03.
*************************************************************
*                                                           *
* This variation sets an error handler for the pending      *
* run unit and then does another check to make sure it      *
* was really set.                                           *
*                                                           *
*************************************************************
     CALL "QLRSETCE" USING new scope newlib old errparm.
     IF oldname IS NOT = "*NONE"
       DISPLAY "** error in oldname "
     END-IF.
     IF newlib IS NOT = "UTCBL"
       DISPLAY "** error in new library "
     END-IF.
*************************************************************
* Call the retrieve API to check to make sure that the      *
* set API worked.                                           *
*************************************************************
     MOVE SPACES TO old exception-id.
     CALL "QLRRTVCE" USING old scope errparm.
     DISPLAY "Retrieved Error Handler is=" old.
     IF oldname IS NOT = "ACERRF24" OR oldlibr IS NOT = "UTCBL"
       DISPLAY "** error - expected ACERRF24 error handler"
     END-IF.
 end-variation.
```

Error Handler for Example COBOL Program

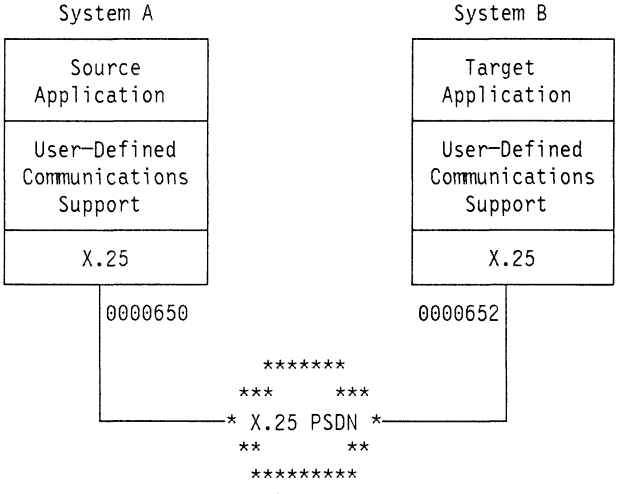
This example error handler works with "Using COBOL Program to Call APIs" on page A-32.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ACERRF24.
************************************************
* Error handler for program ACF24              *
************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-AS400.
 OBJECT-COMPUTER.  IBM-AS400.
 SPECIAL-NAMES. SYSTEM-CONSOLE IS SYSCON.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 scope PIC X VALUE "P".
 01 errparm.
    05 FILLER PIC X(30).
 LINKAGE SECTION.
 77  cobol-id PIC X(7).
 77  valid-responses PIC X(6).
 01  progr.
     05  progname PIC X(10).
     05  proglibr PIC X(10).
 77  system-id PIC X(7).
 77  len-text PIC S9(9) COMP-4.
 01  subtext.
     03  subchars PIC X OCCURS 1 TO 230 TIMES
         DEPENDING ON len-text.
 77  retcode PIC X(1).
 PROCEDURE DIVISION USING cobol-id, valid-responses,
          progr, system-id, subtext, len-text, retcode.
 main-proc.
*************************************************************
* check for typical messages and take appropriate action *
*************************************************************
     EVALUATE cobol-id
     WHEN "LBE7604"
*************************************************************
* stop literal, let the user see the message                *
*************************************************************
       MOVE SPACE TO retcode
     WHEN "LBE7208"
*************************************************************
* accept/display, recoverable problem answer G to continue
*************************************************************
       MOVE "G" TO retcode
     WHEN OTHER
*************************************************************
* for all other messages signal system operator and         *
*  end the current run unit                                 *
*************************************************************
       DISPLAY "COBOL Error Handler ACERRF24 "
               "Found message " cobol-id
               " Issued from program " progr
               UPON syscon
       DISPLAY " Ended current run unit"
               UPON syscon
       MOVE "C" TO retcode
     END-EVALUATE.
     GOBACK.
```

User-Defined Communications Programs for File Transfer

This section provides a simple example showing how X.25-oriented applications use the user-defined communications support to connect to remote systems. Two user-defined application programs written in the C/400 programming language are used to illustrate a simple file transfer between two systems over an X.25 packet-switching data network (PSDN). Although an X.25 example is shown, many of the same concepts can be applied to applications running over token-ring and Ethernet local area networks (LANs). For the purposes of the examples, the APIs are referred to by their call names. For complete information about the user-defined communications APIs, see Chapter 6, "User-Defined Communications Support APIs" on page 6-1.

For this example, the following network configuration will be used.



X.25 Overview

In this example X.25 network, the source application on System A is responsible for establishing a switched virtual circuit, or connection to the target application running on System B. This is done by using the remote network address (System B's address) of X'0000652'. When the target application on System B is initialized, it waits for notification of an incoming call packet before proceeding.

Once the virtual circuit is established, the source application reads records from a file into its output buffer and sends them to the target application using normal X.25 data transfer procedures. While receiving the file data, the target application writes the data to a local file on System B. When the file transfer completes, the source application closes the connection by issuing an X.25 clear request packet and ends. When receiving the clear indication packet, the target application also ends.

User-Defined Communications Support Overview

Both the source and target applications call the Query Line Description (QOLQLIND) API to obtain information about the local X.25 line being used. This information is stored in a local control block for use in establishing the peer connection during X.25 connection processing. Both applications also call the Enable Link (QOLELINK) API to enable the link for future communications. The AS/400 line name, communications handle, and remote DTE address are passed to both programs as arguments to the C function main(). For simplicity, the user space names and data queue name on the call to the QOLELINK API are coded directly in the applications.

Note: Keyed data queue support is used by both applications. The key length is 6 and the keys used are Source and Target for the source and target applications, respectively.

Activating Filters: Once the links have been enabled and both applications have read their respective enable-complete entries from their data queues, the target application program calls the Set Filter (QOLSETF) API to activate a filter. The filter activated then identifies the protocol of the local X.25 service user. This filter is used by the user-defined communications support on System B to route incoming calls. The actual filter type activated is X'00' (for X.25 PID) and its associated value is X'21'. For more information concerning filters, see "Set Filter (QOLSETF) API" on page 6-43. After activating the X'21' filter, the target application waits for the source application to request a connection.

Establishing a Connection: The source application calls the Send Data (QOLSEND) API with a X'B000' operation in its output data buffer to establish a switched virtual circuit (SVC) to the target application. Included in the first byte of the call user data is the protocol ID of the target application, or X'21'. When the user-defined communications support on System B sees the incoming call packet with the first byte of user data equal to a previously activated filter, the call is routed to the process responsible for activating that filter. In this case, the target application will receive notification of an incoming call since it previously activated filter X'21'.

While waiting for the incoming call, the target application calls the Receive Data (QOLRECV) API to receive a X'B201' operation with incoming call data. After doing so, the target application accepts the X.25 connection by calling the QOLSEND API with a X'B400' operation in its output data buffer. See "Send Data (QOLSEND) API" on page 6-27 for more information.

Sending Data: Once the peer connection is established between the source and target applications running on System A and System B respectively, the file transfer takes place. The source application reads records from a local file and calls the QOLSEND API with X'0000' operations in its output data buffer to transfer the file data to System B. This process continues until the entire contents of the source file has been sent to System B.

Receiving Data: After accepting the X.25 connection, the target application waits until its data queue receives incoming-data entries. When the first entry is read from the queue, the QOLRECV API is called to determine which operation was received. Barring failure, the target application should receive a X'0001' operation as a result of the QOLRECV API call. The data contained in the input data buffer is the file data received from System A. While receiving the file data, the target application writes the data to a local file. This process continues until the entire contents of the file is received from System A. The target application then assumes the file transfer is complete when an operation other than a X'0001' operation is received after a successful call to the QOLRECV API. Most likely, the first non-X'0001' operation received will be X'B301' operation, signalling that the user-defined communications support running on System B received an SVC clear indication.

Clearing the Connection and Disabling Links: Once the entire contents of the file has been read and sent to System B, the source application calls the QOLSEND API with a X'B100' operation in its output data buffer to clear the X.25 connection. Afterwards, the source application closes its local file, disables its local link by calling the QOLDLINK API, and ends.

When the source application program sends a X'B100' operation, it causes the target application to receive a X'B301' operation. After receiving this operation, the target application program calls the QOLSEND API with a X'B100' operation to locally close the connection between itself and the user-defined communications support. Afterwards, the target application closes its local file, disables its local link by calling the QOLDLINK API, and ends.

Using Timers and the Data Queue Support: Both the source and target application programs use the user-defined communications support timer service to manage the reception of certain operations. This is done by setting a timer before checking the data queue for an entry. For example, the target application sets a timer to manage the reception of file data from the source application. If the timer expires, the user-defined communications support places a timer-expired entry on the application's data queue. The target application then assumes when receiving this entry that the source application ended abnormally. The target application can then take the appropriate action to end itself.

C/400 Compiler Listings

Below are the listings for the source and target applications described in the previous paragraphs. Note the reference numbers (for example, 1) in the listings. Detailed explanations of each reference number block are found on pages A-55 and A-71.

The target application compiler listing can be found in "Target Application on System B Listing" on page A-56.

Source Application on System A Listing: In this example, the source application is the initiator of all meaningful work. In summary, the source program listed on the following pages does the following:

- Calls the QOLQLIND API to get local X.25 line information
- Opens the local file
- Calls the QOLELINK API to establish a link for communications
- Calls the QOLSEND API with X'B000' operation to establish a peer (SVC) connection
- Sends the local file to the target system via X'0000' operations
- Calls the QOLSEND API with X'B100' operation to clear the peer (SVC) connection
- Calls the QOLDLINK API to disable the link
- Calls the QOLTIMER API to manage the reception of data queue entries

```
Program name  . . . . . : SOURCE
  Library name  . . . . . :     UDCS_APPLS
Source file . . . . . :   QCSRC
  Library name  . . . . . :     UDCS_APPLS
Source member name  . . . . . :   SOURCE
Text Description  . . . . . :   Source Application Example
Compiler options  . . . . . :   *SOURCE  *NOXREF  *SHOWUSR
                              :   *SHOWSYS  *NOSHOWSKP  *NOEXPMAC
                              :   *NOAGR *NOPPONLY  *NODEBUG
                              :   *GEN *NOSECLVL  *PRINT  *LOGMSG
Language level options  . . . . :
Source margins:
  Left margin . . . . . :   1
  Right margin  . . . . . :   80
Sequence columns:
  Left Column . . . . . :
  Right Column  . . . . . :
Define name . . . . . :
Generation options  . . . . . :   *NOLIST *NOXREF *GEN  *NOATR
                              :   *NODUMP  *NOOPTIMIZE *NOALWBND
                              :   *NOANNO
Print file  . . . . . :   QSYSPRT
  Library name  . . . . . :     *LIBL
Message flagging level  . . . . :   0
Compiler message:
  Message limit . . . . . :   *NOMAX
  Message limit severity  . . . :   30
Replace program object  . . . . :   *YES
User profile  . . . . . :   *USER
Authority . . . . . :   *LIBCRTAUT
Target Release  . . . . . :   *CURRENT
INDEBUG options . . . . . :   I don't know
Last change . . . . . :   90/12/19 08:49:37
Source description  . . . . . :   Source Application Example
Compiler  . . . . . :   IBM SAA C/400 Compiler

/*****************************************************************/
/**  Program Name: Source Application Program Example           **/
/**                                                             **/
/**                                                             **/
/**  Function:                                                  **/
/**  This is the source application program example that uses   **/
/**  X.25 services provided by the user-defined communications  **/
/**  support to transfer a simple file to the target application **/
/**  program running on system B.  This program performs the    **/
/**  following:                                                 **/
/**     01.  Open the source file name INFILE.                  **/
/**     02.  Call QOLQLIND API to obtain local line information.**/
/**     03.  Enable a link.                                     **/
/**     04.  Send a 'B000'X operation (call request).           **/
/**     05.  Receive a 'B001'X operation (call confirmation).   **/
/**     06.  Read record(s) from the file opened in step 1). and**/
/**          send '0001'X operation(s) to transfer the file to  **/
/**          the target application program.                    **/
/**     07.  Send a 'B100'X operation (clear call request).     **/
/**     08.  Receive a 'B101'X operation.                       **/
/**     09.  Disable the link enabled in step 3).               **/
/**                                                             **/
/**  A data queue will be actively used to manage the operation **/
/**  of this program.  Data queue support will be used to monitor**/
/**  for the completion of the enable and disable routines, as  **/
/**  well as timer expirations and incoming data.  Timers are   **/
/**  used to ensure that there will never be an infinite wait on**/
```

Figure A-1 (Part 1 of 20). C/400 Compiler Listing for the Source Application

```
/**  the data queue.  If a timer expires, the link enabled will   **/
/**  be disabled and the program will stop.                       **/
/**                                                               **/
/** Inputs:                                                       **/
/** The program expects the following input parameters            **/
/**      Line Name:   This is the name of the line description    **/
/**                   that will be used to call the QOLELINK API. **/
/**                   The line must be an X.25 line with at least **/
/**                   one SVC of type *SVCBOTH or *SVCOUT.        **/
/**                                                               **/
/**      CommHandle:  This is the logical name that will be used  **/
/**                   to identify the link enabled.               **/
/**                                                               **/
/**      Remote DTE Address:  The is the Local Network Address    **/
/**                           of System B.                        **/
/**                                                               **/
/**                                                               **/
/** Outputs:                                                      **/
/** Current status of the file transfer will be provided when     **/
/** running this program.  If an error should occur, then a       **/
/** message will be displayed indicating where the error occurred **/
/** and the program will end.        If the program completes     **/
/** successfully, a "successful completion" message will be       **/
/** posted.                                                       **/
/**                                                               **/
/*******************************************************************/

#include "header"

#include "typedefs"



#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <xxasio.h>
#include <xxcvt.h>
#include <string.h>
#include <ctype.h>
1

typedef struct queuein
    {
      char library??(10??);
      char name??(10??);
      char option;
    } queuein;

typedef struct namelib
    {
      char library??(10??);
      char name??(10??);
    } namelib;

typedef _Packed struct format1
  {
   char type;
   char reserved1;
   unsigned short logchanid;
```

Figure A-1 (Part 2 of 20). C/400 Compiler Listing for the Source Application

```
   unsigned short sendpacksize;
   unsigned short sendwindsize;
   unsigned short recvpacksize;
   unsigned short recvwindsize;
   char reserved2??(7??);
   char dtelength;
   char dte??(16??);
   char reserved3??(8??);
   char dbit;
   char reserved4??(7??);
   char cug;
   char cugid;
   char reverse;
   char fast;
   char faclength;
   char facilities??(109??);
   char reserved5??(48??);
   unsigned short calllength;
   char callud??(128??);
   char reserved6??(128??);
   unsigned char misc??(4??);          /* control flags */
   unsigned int maxasmsize;
   unsigned short autoflow;
} format1;

typedef _Packed struct format2
  {
   unsigned short type;
   char cause;
   char diagnostic;
   char reserved??(4??);
   char faclength;
   char facilities??(109??);
   char reserved2??(48??);
   unsigned short length;
   char userdata??(128??);
  } format2;

typedef _Packed struct desc
  {
   unsigned short length;
   char more;   /*These 4 char's are only used for X.25.*/
   char qualified;
   char interrupt;
   char dbit;
   char reserved??(26??);
  } desc;

typedef _Packed struct llcheader
  {
   unsigned short headerlength;
   char macaddr??(6??);
   char dsap;
   char ssap;
   char priority;
   char priorctl;
   unsigned short routlen;
   unsigned short userdtalen;
   char data??(1??);
  } llcheader;
```

Figure A-1 (Part 3 of 20). C/400 Compiler Listing for the Source Application

```
typedef _Packed struct espec
  {
   char reserved??(2??);
   unsigned int hwecode;
   unsigned int timestamphi;
   unsigned int timestamplo;
   unsigned int elogid;
   char reserved2??(10??);
   char flags;
   char cause;
   char diagnostic;
   char reserved3;
   unsigned int erroroffset;
   char reserved4??(4??);
  } espec;

typedef struct tableentry
  {
    char handle??(10??);
    char type;
    char inbuff??(20??);
    char indesc??(20??);
    char outbuff??(20??);
    char outdesc??(20??);
    unsigned int totaldusize;
    struct tableentry *next;
  } tableentry;

/*******  Data structure for X.25 line    ********/
/**** descriptions as returned by QOLQLIND. ******/

typedef struct x25info
  {
   char addrlen;
   char addr??(9??);
   char addrtype;
   char insert;
   char modulus;
   char dtedce;
   unsigned short maxsend;
   unsigned short maxrecv;
   unsigned short defsend;
   unsigned short defrecv;
   char windowsend;
   char windowrecv;
   unsigned short numlc;
   char lcinfo??(4??);
  } x25info;

typedef struct querydata
  {
   char header??(12??);    /* line header info */
   x25info x25data;        /* preliminary data */
  } querydata;
#include "hexconv"
#include <stdio.h>

char *inttohex(decimal,hex)  /*Converts a 4-byte integer
       into a string of 2 uppercase hex characters.*/
unsigned int decimal;
char *hex;
```

Figure A-1 (Part 4 of 20). C/400 Compiler Listing for the Source Application

```
{
  sprintf(hex,"%.2X",decimal);
  return(hex);
}

unsigned int hextoint(hex)  /*Converts a string containing
               hex digits into a 4-byte integer.   */
char *hex;
{
  int decimal;

  sscanf(hex,"%x",&decimal);
  return(decimal);
}
```
2
```
#pragma linkage(QOLDLINK, OS)
#pragma linkage(QOLELINK, OS)
#pragma linkage(QOLSEND, OS)
#pragma linkage(QOLRECV, OS)
#pragma linkage(QUSPTRUS, OS)
#pragma linkage(QRCVDTAQ, OS)
#pragma linkage(QCLRDTAQ, OS)
#pragma linkage(QOLTIMER, OS)
#pragma linkage(QOLSETF, OS)
#pragma linkage(QOLQLIND, OS)

FILE *screen;
FILE *rptr;
FILE *fptr;

extern void QOLDLINK(int *, int *, char *, char *);
```
3
```
extern void QOLELINK (int *, int *, int *, int *, int *, int *,\
                      char *, char *, char *, char *, int *, char *,\
                       char *, char *, char *);

extern void QOLSEND (int *, int *, void *, int *, int *, int *,\
                      char *, unsigned short *, int *);

extern void QOLRECV (int *, int *, int *, int *, unsigned short *,\
                      int *, char *, void *, char *);

extern void QOLSETF (int *, int *, int *, char *);

extern void QOLTIMER (int *, int *, char *, char *, char *, char *,
                       int *, int *, int *, char *, char *);

extern void QUSPTRUS (void *, void *);

extern void QRCVDTAQ (char *, char *, char *, void *, char *,\
                       char *, char *, char *, char *, char *);

extern void QCLRDTAQ (char *, char *);

extern void QOLQLIND(int *, int *, int *, void *, char *, char *);

/************   Typedef  Declarations      ******************/

typedef struct usrspace
    {
      char name??(10??);
      char library??(10??);
    } usrspace;
```

Figure A-1 (Part 5 of 20). C/400 Compiler Listing for the Source Application

4

```
typedef struct enableparms     /*  Enable parameters  */
  {
   int retcode,              /*  Output  */
       reason,               /*  Output  */
       tdusize,              /*  Output  */
       numunits,             /*  Output  */
       maxdtalan,            /*  Output  */
       maxdtax25,            /*  Input   */
       keylength;            /*  Input   */
char keyvalue??(256??),      /*  Input  */
     linename??(10??);       /*  Input  */
  } enableparms;

typedef struct disableparms     /*  Disable parameters  */
  {
  int retcode,              /*  Output  */
      reason;               /*  Output  */
  char vary;                /*  Input  */
  } disableparms;

typedef struct setfparms   /*  Set Filters parameters  */
  {
  int retcode,              /*  Output  */
      reason,               /*  Output  */
      erroffset;            /*  Output  */
  } setfparms;

typedef _Packed struct hdrparms /* Filter header */
  {
   char function;
   char type;
   unsigned short number;
   unsigned short length;
   char filters??(1??);
  } hdrparms;

typedef _Packed struct x25filter /* X.25 filter */
  {
   char pidlength;
   char pid;
   char dtelength;
   char dte??(12??);
   char flags;
  } x25filter;

typedef struct sendparms     /*  Send parameters  */
  {
 espec errorspecific;       /*  Output  */
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       newpcep,             /*  Output  */
       ucep,                /*  Input  */
       pcep,                /*  Input   */
       numdtaelmnts;        /*  Input   */
unsigned short operation;   /*  Input  */
  } sendparms;
```

Figure A-1 (Part 6 of 20). CI400 Compiler Listing for the Source Application

```
typedef struct recvparms   /*  Receive parameters  */
  {
 espec errorspecific;      /*  Output  */
   int retcode,            /*  Output  */
       reason,             /*  Output  */
       newpcep,            /*  Output  */
       ucep,               /*  Output  */
       pcep,               /*  Output   */
       numdtaunits;        /*  Output   */
  char dataavail;          /*  Output  */
unsigned short operation;  /*  Output  */
  } recvparms;

typedef struct timerparms  /*  Timer parameters  */
  {
   int retcode,            /*  Output  */
       reason,             /*  Output  */
       interval,           /*  Input  */
       establishcount,     /*  Input  */
       keylength;          /*  Input   */
  char handleout??(8??),   /*  Output  */
       handlein??(8??),    /*  Input  */
       operation,          /*  Input  */
       keyvalue??(256??),  /*  Input  */
       userdata??(60??);   /*  Input  */
  } timerparms;


typedef struct especout
  {
   char hwecode??(8??);
   char timestamp??(16??);
   char elogid??(8??);
   char fail;
   char zerocodes;
   char qsysopr;
   char cause??(2??);
   char diagnostic??(2??);
   char erroffset??(6??);
  } especout;

typedef struct qlindparms     /*  Query line parameter
  {
   int retcode,            /*  Output  */
       reason,             /*  Output  */
       nbytes;             /*  Output  */
   char userbuffer??(256??);
   char format;
  } qlindparms;

typedef _Packed union content       /*  Queue support
  {
    _Packed struct other
      {
       char commhandle??(10??);
       char reserved??(58??);
      } other;
```

Figure A-1 (Part 7 of 20). C/400 Compiler Listing for the Source Application

```
    _Packed struct enable
      {
       char commhandle??(10??);
       char status;
       char reserved??(57??);
      } enable;
    _Packed struct timer
      {
       char timerhandle??(8??);
       char userdata??(60??);
      } timer;
  } content;

typedef _Packed struct qentry  /*  Queue  parameter */
  {
    char type??(10??);
    unsigned short msgid;
    content message;
    char key??(256??);
  } qentry;
```

5

```
void senddata(sendparms *a, char *b, desc *c, char *d, char *e, int f);
void sndformat1(sendparms *a,char *b, char *c, char *d, qlindparms *f);
void sndformat2 (sendparms *a, char *b, char *c);
void setfilters (hdrparms *a);
void byte (char *a, int b, char *c, int d);
void printespec (espec *a);
void settimer(unsigned short *a,char *b,qentry *c,usrspace *d,char *e);
void dequeue (int a, char *b, qentry *c, usrspace *d);
void x25lind (qlindparms *a, char *b);
int getline (char *a, int b, FILE *c);
void disablelink (disableparms *a, char *b, usrspace *c);
void handler (disableparms a, usrspace *b);
sigdata_t *sigdata(void);

/*************************************************************/
/***************   Start Main Program   *******************/
/*************************************************************/

main (int argc, char *argv??(??))
{

/************ Variable  Declarations      *******************/

  usrspace inbuff,       /*  Input Data Buffer */
           indesc,       /*  Input Buffer Descriptor  */
           outbuff,      /*  Output Data Buffer  */
           outdesc,      /*  Output Buffer Descriptor */
           qname;        /*  Data Queue  */

  int length,                 /* Data Queue key length  */
      linesiz,                /* Length of line that is read in */
      i= 0;                   /* counter  */
  unsigned short expctid;     /*  Message ID that is expected */
  char commhandle??(10??),    /*  Command Line Parameter  */
       *buffer,               /* Pointer to buffer     */
       rmtdte??(18??),        /* Remote DTE read in       */
       line??(132??),         /* Line to read in       */
       key??(256??);          /* Data Queue key identifier  */
  desc *descriptor;           /* Pointer to buffer descriptor  */
```

Figure A-1 (Part 8 of 20). CI400 Compiler Listing for the Source Application

```
/** definitions for the API functions  **/
enableparms enable;
disableparms disable;
sendparms send;
recvparms recv;
setfparms setf;
timerparms timer;
qlindparms qlind;
qentry dataq;
hdrparms *header;
```

6

```
/***---  Open the file to send to remote side        ----**/
if ((fptr = fopen("UDCS_APPLS/INFILE(INFILE))", "r")) == N
   {
   printf("Unable to open source input file in UDCS_APPLS LIB.\n"
   printf("The Program was terminated.\n\n");
   return;
   }
/***---  Open the display file as our input screen.  ----**/
if ((screen = fopen("ERRORSPEC", "ab+ type=record")) == NULL)
   {
   printf("Unable to open display file.\n");
   printf("The Program was terminated.\n\n");
   return;
   }

/** set the exception handler  **/
signal(SIGABRT,&handler);

/**  Clear the command line Parameters  **/
strncpy(enable.linename, "          ", 10);  /* Clear linename */
strncpy(commhandle, "          ", 10);       /* Clear Commhandle*/
strncpy(rmtdte, "                ", 17);     /* Clear Remote DTE*/

/**  Receive command line Parameters  **/
strncpy(enable.linename, argv??(1??), strlen(argv??(1??)));
strncpy(commhandle, argv??(2??), strlen(argv??(2??)));
strncpy(rmtdte, argv??(3??), strlen(argv??(3??)));
rmtdte??(strlen(argv??(3??))??) = '\0';

/** Initialize the user spaces  **/
strncpy(inbuff.library, "UDCS_APPLS", 10);    /*  Input Buffer */
strncpy(inbuff.name, "SOURCEIBUF", 10);
strncpy(indesc.library, "UDCS_APPLS", 10);    /* Input B Desc  */
strncpy(indesc.name, "SOURCEBDSC", 10);
strncpy(outbuff.library, "UDCS_APPLS", 10);   /*  Output Buffer*/
strncpy(outbuff.name, "SOURCEOBUF", 10);
strncpy(outdesc.library, "UDCS_APPLS", 10);   /* Output B Desc */
strncpy(outdesc.name, "SOURCEODSC", 10);
strncpy(qname.library, "UDCS_APPLS", 10);   /*  Data queue  */
strncpy(qname.name, "X25DTAQ   ", 10);

/*****   retrieve the line description information  ******/
x25lind (&qlind, enable.linename);
```

Figure A-1 (Part 9 of 20). C/400 Compiler Listing for the Source Application

```
  if ((qlind.retcode != 0) || (qlind.reason != 0))
     {
     printf("Query line description failed.\n");
     printf("Return code = %d\n", qlind.retcode);
     printf("Reason code = %d\n\n", qlind.reason);
     return;
     }

  /*****   Hard Code the QOLELINK Input Parameters  ******/
  enable.maxdtax25 = 512;
  enable.keylength = 3;
  strncpy (enable.keyvalue, "SND", 3);
7

  /************************************************/
  /************  Enable the line  *****************/
  /************************************************/
  QOLELINK (&(enable.retcode), &(enable.reason), &(enable.tdusize),\
      &(enable.numunits), &(enable.maxdtalan), &(enable.maxdtax25),\
      (char *)&inbuff, (char *)&indesc, (char *)&outbuff,\
      (char *)&outdesc, &(enable.keylength), enable.keyvalue,\
      (char *)&qname, enable.linename, commhandle);

  if ((enable.retcode != 0) || (enable.reason != 0))
     {
     printf("Line %.10s with Commhandle %.10s was NOT ENABLED.\n",\
             enable.linename, commhandle);
     printf("Return code = %d\n", enable.retcode);
     printf("Reason code = %d\n\n", enable.reason);
     return;
     }
8

  /*-------- Set a timer for Enable Link  ---------**/
  expctid = 0xF0F0;
  settimer(&expctid, "Enable", &dataq, &qname, commhandle);
  if (expctid != 0xF0F0)
     {
     disablelink (&disable, commhandle, &qname);
     return;
     }

9

  /*********************************************************************/
  /**************   Set up a Call Request Packet  ******************/
 /*********************************************************************/


  /****   Get pointers to the user spaces.   ******/
  QUSPTRUS(&outbuff, &buffer);
  QUSPTRUS(&outdesc, &descriptor);

  send.ucep = 26;              /*  set the UCEP number  */
  send.operation = 0xB000;     /*  send a call request  */
  send.numdtaelmnts = 1;       /*  send one data unit   */

  /**-----------    Send the packet   ---------**/
  sndformat1 (&send, buffer, rmtdte, commhandle, &qlind);
```

Figure A-1 (Part 10 of 20). C/400 Compiler Listing for the Source Application

```
if ((send.retcode != 0) || (send.reason != 0))
   {
   printf("Call request packet not sent\n");
   printf("Return code = %d\n", send.retcode);
   printf("Reason code = %d\n", send.reason);
   printf("new pcep %d\n", send.newpcep);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }
```

10

```
/*****************************************************************/
/***********   Receive the Call CONFIRMATION packet       ********/
/*****************************************************************/

/*-------- Set a timer to receive a message  ---------**/
expctid = 0xF0F3;
settimer(&expctid, "Rcv Call", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }

/** Get pointer to use space  **/
QUSPTRUS (&inbuff, &buffer);
QUSPTRUS (&indesc, &descriptor);

QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
         &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
         &(recv.dataavail), &(recv.errorspecific), commhandle);

if ((recv.retcode != 0) || (recv.reason != 0))
   {
   printf("Recv Call reqst resp failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

/* Interpret the Received Operation  */
if (recv.operation != 0xB001)
   {
   printf("Recvd opr %x instead of opr B001\n", recv.operation);
   disablelink (&disable, commhandle, &qname);
   return;
   }

printf("We have an X.25 SVC connection\n\n");
```

Figure A-1 (Part 11 of 20). C/400 Compiler Listing for the Source Application

11
```
/*****************************************************************/
/***************   Send the file to the target application *******/
/*****************************************************************/

send.pcep = send.newpcep;       /*  set the PCEP number  */

/***************   Send the Mbr LGRF in file DOC  ****************/
linesiz = getline(line, 92, fptr);   /* Get first record  **/
while (linesiz != 0)
   {
   /***************   Send a Packet of Data    ***************/

   /****   Get pointers to the user spaces.   ******/
   QUSPTRUS(&outbuff, &buffer);
   QUSPTRUS(&outdesc, &descriptor);

   send.operation = 0x0000;
   send.numdtaelmnts = 1;

   /**-----   Send the packet     ------------**/
   senddata (&send, buffer, descriptor, commhandle, line, linesiz);

   if ((send.retcode != 0) || (send.reason != 0))
      {
      printf("Data NOT sent for commhandle %.9s\n", commhandle);
      printf("Return code = %d\n", send.retcode);
      printf("Reason code = %d\n", send.reason);
      printf("new pcep %d\n", send.newpcep);
      printespec(&(send.errorspecific));

      disablelink (&disable, commhandle, &qname);
      return;
      }
   i = i + 1;
   printf("Data %d Sent for commhandle %.9s.\n\n", i, commhandle);

   linesiz = getline(line, 92, fptr);  /** Get next record  **/
   }                        /***  End While loop  ***/

/*****************************************************************/
/***************   Set up a Clear Request Packet  **************/
/*****************************************************************/
```
12
```
/****   Get pointers to the user spaces.   ******/
QUSPTRUS(&outbuff, &buffer);
QUSPTRUS(&outdesc, &descriptor);

send.operation = 0xB100;         /**  send clear request  **/
send.numdtaelmnts = 1;           /**  send one data unit  **/

/**-----------     Send the packet   ---------**/
sndformat2 (&send, buffer, commhandle);

if ((send.retcode != 0) || (send.reason != 0))
   {
   printf("Clear request packet not sent\n");
   printf("Return code = %d\n", send.retcode);
   printf("Reason code = %d\n", send.reason);
   printf("new pcep %d\n", send.newpcep);
   printespec(&(send.errorspecific));
```

Figure A-1 (Part 12 of 20). CI400 Compiler Listing for the Source Application

```
    disablelink (&disable, commhandle, &qname);
    return;
    }
```

13

```
  /*****************************************************************/
  /***********   Receive the Clear Request Response packet   *****/
  /*****************************************************************/

  /*-------- Set a timer to receive a message  ---------**/
  expctid = 0xF0F3;
  settimer(&expctid, "Rv Clr Rqt", &dataq, &qname, commhandle);
  if (expctid != 0xF0F3)
     {
     disablelink (&disable, commhandle, &qname);
     return;
     }


  /***********   Call QOLRECV to Receive the Clear Response  *****/
  /****   Get pointers to the user spaces.   ******/
  QUSPTRUS (&inbuff, &buffer);
  QUSPTRUS (&indesc, &descriptor);

  QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
           &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
           &(recv.dataavail), &(recv.errorspecific), commhandle);

  if ((recv.retcode != 0) || (recv.reason != 0))
     {
     printf("Recv clear response failed\n");
     printf("return code %d\n", recv.retcode);
     printf("reason code %d\n", recv.reason);
     printespec(&(send.errorspecific));

     disablelink (&disable, commhandle, &qname);
     return;
     }

  /* Interpret the Received Operation  */
  if (recv.operation != 0xB101)
     {
     printf("Recvd opr %x instead of opr B101\n", recv.operation);
     disablelink (&disable, commhandle, &qname);
     return;
     }

  /*********************************************/
  /***  Disable the link and end the program  ****/
  /*********************************************/
  disablelink (&disable, commhandle, &qname);

  printf("******   SSNDTH completed successfully   ******\n\n");

}    /* End Main */

/*******************************************************************/
/************** Start Subroutine Section    **********************/
/*******************************************************************/
```

Figure A-1 (Part 13 of 20). C/400 Compiler Listing for the Source Application

```
/*************************************************************/
/***************     Send a Packet of Data  ******************/
 14

void senddata (sendparms *send,
               char *buffer,
               desc *descriptor,
               char *commhandle,
               char *line,
               int linesiz)

{
  descriptor->length = linesiz;
  descriptor->more = 0;
  descriptor->qualified = 0;
  descriptor->interrupt = 0;
  descriptor->dbit = 0;
  strncpy (buffer, line, linesiz);

  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep), \
           commhandle, &(send->operation), &(send->numdtaelmnts));

}  /* End senddata Subroutine  */


/*********************************************************/
/************  Routine to fill X.25 Format I   ***********/

void sndformat1 (sendparms *send,
                 char *buffer,
                 char *rmtdte,
                 char *commhandle,
                 qlindparms *qlind)

{
  format1 *output = (format1 *) buffer;
  register int counter;
  register querydata *qd;

  qd = (querydata *)&(qlind->userbuffer);
  output->type = 2;                 /*  SVC used  */
  output->logchanid = 0x0;
  output->sendpacksize = qd->x25data.defsend;
  output->sendwindsize = qd->x25data.windowsend;
  output->recvpacksize = qd->x25data.defrecv;
  output->recvwindsize = qd->x25data.windowrecv;

  output->dtelength = strlen(rmtdte);
  byte(output->dte, 16, rmtdte, strlen(rmtdte));
  output->dbit = 0;
  output->cug = 0;
  output->cugid = 0;
  output->reverse = 0;
  output->fast = 0;
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->calllength = 1;
  byte(output->callud, 128, "21", 2);   /* Contains Remote PID */
```

Figure A-1 (Part 14 of 20). C/400 Compiler Listing for the Source Application

```
  output->misc??(0??) = 0;     /* change to 0x80 for reset support */
  output->misc??(1??) = 0;
  output->misc??(2??) = 0;
  output->misc??(3??) = 0;
  output->maxasmsize = 16383;
  output->autoflow = 32;


  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));

}  /* End sndformat1 Subroutine  */


/*********************************************************/
/************  Routine to fill X.25 Format II   ***********/

void sndformat2 (sendparms *send,
                 char *buffer,
                 char *commhandle)

{
  format2 *output = (format2 *) buffer;

  output->type = 1;
  output->cause = 'FF';
  output->diagnostic = 'FF';
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->length = 0;
  byte(output->userdata, 128, "", 0);


  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));

}  /* End sndformat2 Subroutine  */
```

15

```
/*****************************************/
/********    Routine to disable   ***********/

void disablelink (disableparms *disable,
                  char *commhandle,
                  usrspace *qname)

{
unsigned short expctid;
qentry dataq;

  disable->vary = 1;     /*  Hard coded to be varied off  */

  QOLDLINK (&(disable->retcode), &(disable->reason),\
                        commhandle, &(disable->vary));
```

Figure A-1 (Part 15 of 20). C/400 Compiler Listing for the Source Application

```
  if ((disable->retcode != 0) && (disable->reason != 00))
     {
     printf ("Link %.10s did not disabled.\n", commhandle);
     printf ("return code = %d\n", disable->retcode);
     printf ("reason code = %d\n\n", disable->reason);
     }

  /**-------  Set a timer to receive disable complete msg --------**/
  expctid = 0xF0F1;
  settimer(&expctid, "Disable", &dataq,  qname, commhandle);
  if (expctid != 0xF0F1)
     {
     printf("Disable link did not complete successfully");
     return;
     }

  printf ("%.10s link disabled \n", commhandle);

  /**  close the files  **/
  fclose(fptr);
  fclose(screen);

}  /* End disablelink Subroutine  */


/*************************************************************/
/**    Routine to convert string to Hexadecimal format   ******/

void byte (char *dest,
           int dlength,
           char *source,
           int slength)

{
  register int counter;
  char holder??(2??);

  for (counter=0;counter<dlength;counter++)
   dest??(counter??)=0;
  for (counter=slength-1;counter>=0;counter--)
   if isxdigit(source??(counter??))
     {
      holder??(0??)=source??(counter??);
      holder??(1??)='\0';
      if  (counter % 2 == 0)
        dest??(counter/2??) += (char) hextoint(holder)*16;
      else dest??(counter/2??) += (char) hextoint(holder);
     }

}  /* End byte Subroutine  */

/*************************************************************/
/**    Routine to display the ErrorSpecific output       ******/

void printespec(espec *errorspecific)

{
  especout outparms;
```

Figure A-1 (Part 16 of 20). C/400 Compiler Listing for the Source Application

```
  QXXFORMAT(screen, "ERRORSPEC ");
  sprintf(outparms.hwecode, "%.8X", errorspecific->hwecode);
  sprintf(outparms.timestamp, "%.8X%.8X", errorspecific->timestamphi,\
                                    errorspecific->timestamplo);
  sprintf(outparms.elogid, "%.8X", errorspecific->elogid);
  if (errorspecific->flags & 0x40)
    outparms.fail = 'Y';
  else outparms.fail = 'N';
  if (errorspecific->flags & 0x20)
    outparms.zerocodes = 'Y';
  else outparms.zerocodes = 'N';
  if (errorspecific->flags & 0x10)
    outparms.qsysopr = 'Y';
  else outparms.qsysopr = 'N';
  sprintf(outparms.cause,"%.2X", errorspecific->cause);
  sprintf(outparms.diagnostic, "%.2X", errorspecific->diagnostic);
  sprintf(outparms.erroffset, "%.6d", errorspecific->erroroffset);
  fwrite(&outparms, 1, sizeof(especout), screen);
  fread("", 0, 0, screen);

}  /* End printespec Subroutine  */

/*************   Set a timer and dequeue next entry  ******/
16

void settimer (unsigned short *expctid,
               char *process,
               qentry *dataq,
               usrspace *qname,
               char *commhandle)

{
timerparms timer;
disableparms disable;
int length;
char key??(6??);

  timer.interval = 20000;      /*  Set timer for 20 seconds */
  timer.establishcount = 1;
  timer.keylength = 6;          /*  No key value  */
  strncpy(timer.keyvalue, "SOURCE", 6);
  timer.operation = 1;           /*  Set a timer  */

  QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
          timer.handlein, (char *)qname, &(timer.operation),\
          &(timer.interval), &(timer.establishcount),\
          &(timer.keylength), timer.keyvalue, timer.userdata);

  if ((timer.retcode != 0) || (timer.reason != 0))
     {
     printf("%s timer failed while being set.\n", process);
     printf("Return code = %d\n", timer.retcode);
     printf("Reason code = %d\n\n", timer.reason);
     }
```

Figure A-1 (Part 17 of 20). C/400 Compiler Listing for the Source Application

```
/**-------  Dequeue an entry  --------**/
  strncpy(key, "SOURCE",6);
  length = 6;
  dequeue (length, key, dataq, qname);

  /*** Cancel timer  ***/
  if (dataq->msgid != 0xF0F4)
     {
     strncpy(timer.handlein, timer.handleout, 8);
     timer.operation = 2;           /*  Set one timer  */

     QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
           timer.handlein, (char *)qname, &(timer.operation),\
           &(timer.interval), &(timer.establishcount),\
           &(timer.keylength), timer.keyvalue, timer.userdata);

     if ((timer.retcode != 0) || (timer.reason != 0))
        {
        printf("%s timer failed while being canceled\n", process);
        printf("Return code = %d\n", timer.retcode);
        printf("Reason code = %d\n\n", timer.reason);
        }
     }

  if (dataq->msgid != *expctid)
      {
      printf ("A %.4X message ID was received instead of %.4X\n",\
              dataq->msgid, *expctid);
      printf ("%s completion message was not received\n", process);
      *expctid = dataq->msgid;
      }

}  /* End settimer Subroutine  */

/*****************************************************************/
/*******   Dequeues the Incoming Message and processes it ******/

void dequeue (int length,
              char *key,
              qentry *dataq,
              usrspace *qname)

{
  char fldlen??(3??),
       waittime??(3??),
       keylen??(2??),
       senderid??(2??),
       *pointer,
       order??(2??);
  register int counter;


  waittime??(0??) = 0;
  waittime??(1??) = 0;
  waittime??(2??) = 0x1D;   /*  Hard code a delay of infinite   */
  keylen??(0??) = 0;
  keylen??(1??) = 0x6F;   /* Hard code a keylength of 6  */
  senderid??(0??) = 0;
  senderid??(1??) = 0x0F;
  strncpy(order, "EQ", 2);
```

Figure A-1 (Part 18 of 20). C/400 Compiler Listing for the Source Application

```
  fflush(stdin);
  pointer = (char *)dataq;
  for (counter = 0; counter < 336; counter++)
    pointer??(counter??) = 0;

  strncpy (dataq->type, "       ", 7);
  while ((strncmp(dataq->type, "*USRDFN", 7) != 0) || (fldlen == 0))
     QRCVDTAQ(qname->name, qname->library, fldlen, dataq, waittime,\
           order, keylen, key, senderid,"");

}  /* End dequeue Subroutine  */


/*************************************************************/
/**  x25lind:  Read a record into buf and return length   **/
```

17

```
void x25lind (qlindparms *qlind, char *linename)
{
register int counter;

  for(counter=0;counter<256;counter++)
     qlind->userbuffer??(counter??)=0;

  qlind->format = 0x01;
  QOLQLIND (&(qlind->retcode), &(qlind->reason), &(qlind->nbytes),\
            qlind->userbuffer, linename, &(qlind->format));

}  /*  End x25lind Subroutine  */

/*****************************************************/
/**  Getline:  Read a record into line and return length   **/

int getline (char *line, int max, FILE *fptr)
{
   if (fgets(line, max, fptr) == NULL)
      return 0;
   else
      return strlen(line);

}  /* End getline Subroutine  */

/*************************************************/
/*  Exception handler, so that a failure will not
    kill the program, and any associated data!!  */
```

18

```
void handler (disableparms disable, usrspace *qname)
{
  sigdata_t *data;

  disablelink(&disable, "*ALL      ", qname);
  printf("The program received an excecption.\n");
  printf("Disable Link was called & the program was terminated.\n\n");

  data=sigdata();
  data->sigact->xhalt=0;
  data->sigact->xrtntosgnler=0;
  data->sigact->xresigprior=0;
  data->sigact->xresigouter=0;

}  /*  End handler Subroutine  */

         * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure A-1 (Part 19 of 20). C/400 Compiler Listing for the Source Application

```
            * * * * *  I N C L U D E S  * * * * *
Include Name        Last change        Actual Include Name
header              90/12/19 08:49:31  UDCS_APPLS/QCSRC/HEADER
typedefs            90/12/19 08:49:31  UDCS_APPLS/QCSRC/TYPEDEFS
stdio.h             90/11/12 17:24:55  QCC/H/STDIO
stddef.h            90/11/12 17:24:54  QCC/H/STDDEF
errno.h             90/11/12 17:24:49  QCC/H/ERRNO
signal.h            90/11/12 17:24:53  QCC/H/SIGNAL
ctype.h             90/11/12 17:24:49  QCC/H/CTYPE
stdarg.h            90/11/12 17:24:54  QCC/H/STDARG
stdlib.h            90/11/12 17:24:56  QCC/H/STDLIB
signal.h            90/11/12 17:24:53  QCC/H/SIGNAL
xxasio.h            90/11/12 17:24:57  QCC/H/XXASIO
xxcvt.h             90/11/12 17:24:58  QCC/H/XXCVT
string.h            90/11/12 17:24:56  QCC/H/STRING
ctype.h             90/11/12 17:24:49  QCC/H/CTYPE
hexconv             90/12/19 08:49:27  UDCS_APPLS/QCSRC/HEXCONV
stdio.h             90/11/12 17:24:55  QCC/H/STDIO
          * * * * *  E N D  O F  I N C L U D E S  * * * * *

           * *  M E S S A G E  S U M M A R Y  * *
 Total   Info    Warning    Error    Severe    Terminal
       (0-4)    (5-19)    (20-29)   (30-39)    (40-99)
   0       0         0          0         0          0
      * *  E N D  O F  M E S S A G E  S U M M A R Y  * *


    IBM SAA C/400   UDCS_APPLS/SOURCE
 ROUTINE          BLOCK NUMBER  SCOPE  TYPE
 <MAIN>                  2      LOCAL  MAIN-PROGRAM
 __sigdata               6      LOCAL  PROCEDURE
 __sgnl                 12      LOCAL  PROCEDURE
 __frdinit              49      LOCAL  PROCEDURE
 __getrec               50      LOCAL  PROCEDURE
 __putrec               51      LOCAL  PROCEDURE
 __frdexit              52      LOCAL  PROCEDURE
 __fwrinit              53      LOCAL  PROCEDURE
 __fwrexit              54      LOCAL  PROCEDURE
 QXXFORMAT             129      LOCAL  PROCEDURE
 __strlen              164      LOCAL  PROCEDURE
 __strncmp             168      LOCAL  PROCEDURE
 __strncpy             170      LOCAL  PROCEDURE
 inttohex              181      ENTRY   PROCEDURE
 hextoint              182      ENTRY   PROCEDURE
 senddata              196      ENTRY   PROCEDURE
 sndformat1            197      ENTRY   PROCEDURE
 sndformat2            198      ENTRY   PROCEDURE
 byte                  200      ENTRY   PROCEDURE
 printespec            201      ENTRY   PROCEDURE
 settimer              202      ENTRY   PROCEDURE
 dequeue               203      ENTRY   PROCEDURE
 x25lind               204      ENTRY   PROCEDURE
 getline               205      ENTRY   PROCEDURE
 disablelink           206      ENTRY   PROCEDURE
 handler               207      ENTRY   PROCEDURE
 main                  208      ENTRY   PROCEDURE
* * * * *  E N D  O F  C O M P I L A T I O N  * *
```

Figure A-1 (Part 20 of 20). C/400 Compiler Listing for the Source Application

The following reference numbers and explanations correspond to the reference numbers in the source application's program listing.

1 Some general C structure declarations used by both the source and target application programs.

Note: This example program was developed using a 5250 display station and keyboard where the left ([) and right (]) bracket characters are not supported. As a result of this, C language array declarations used in the applications use the character sequence of "??(" to denote a left bracket and "??)" to denote a right bracket.

You do not need to change C language applications containing array declarations with the bracket characters in order to compile and run on the AS/400.

2 C/400 compiler directives are used here to indicate that standard OS/400 linkage conventions should be used when calling the user-defined communications support APIs.

3 C external function definitions for the user-defined communications support APIs. Note that all parameters are passed by reference.

4 More C structure declarations that are used when calling the user-defined communications support APIs.

5 Function prototypes of the internal functions used in this program.

6 Call the C library routines fopen() and signal() to open the source file and set up a signal handler to process AS/400 exceptions, respectively. An example of an exception would be accessing a data area with a NULL pointer. If an exception situation is encountered, the handler() will be called in order for the program to end.

7 Call the QOLQLIND API to retrieve local configuration information from the AS/400 line description about that will be used for communications. Next, call the QOLELINK API to enable the line description using the line name and communications handle passed as input parameters to this program.

8 Call the QOLTIMER API to time the completion of the enable link operation. If the timer expires before the enable-complete entry is posted on the this program's data queue, then this program will end.

9 Call the QOLSEND API with a X'B000' operation to establish a connection to the target application program.

10 Monitor the source program's data queue for the call confirmation. The source program will be notified of the call confirmation by call the QOLRECV API and receiving a X'B001' operation in the program's input buffer.

11 This is the main send loop for the source program. The data from the source file is placed one line at a time in the output buffer and then the QOLSEND API is called to send one data unit of the file to System B. This process repeats until the contents of the entire file have been transmitted to the target application.

12 Call the QOLSEND API with a X'B100' operation to clear the peer connection.

13 The source program will check its data queue for a response to the clear packet sent to the target system. Once the response is received, the program will clean up, call the QOLDLINK API to disable the link previously enabled, and end.

14 The following C functions illustrate the various user-defined communications support APIs.

15 This procedure illustrates a call to the QOLDLINK API. Note the vary option is set to vary off the associated AS/400 *USRDFN network device.

16 The settimer() calls the QOLTIMER API requesting timers for 20000 milliseconds, or twenty seconds. After setting a timer, the settimer() will call the dequeue() to remove an entry from the program's data queue.

17 The x25lind() illustrates calling the QOLQLIND API.

18 As mentioned in block 6, the handler() will be called when OS/400 exception situation is encountered. This function performs final processing, calls the QOLDLINK API, and ends the source application program.

Target Application on System B Listing: The target application waits for the source application to initiate the file transfer. The following list summarizes the actions of the target application:

- Calls the QOLQLIND API to get local X.25 line information
- Opens the local file
- Calls the QOLELINK API to establish a link for communications
- Calls the QOLSETF API to activate an X.25 protocol ID filter
- Calls the QOLRECV API to receive the X'B201' operation (incoming call)
- Calls the QOLSEND API with a X'B400' operation to accept the SVC connection
- Receives the file from the target system via X'0001' operations
- Calls the QOLRECV API to receive the X'B301' (connection failure notification)
- Call the QOLSEND API with 'B100' operation to locally close the SVC connection
- Calls the QOLDLINK API to disable the link
- Calls the QOLTIMER API to manage the reception of data queue entries

Explanations of the reference numbers in the listing can be found on page A-71.

```
Program name . . . . . : TARGET
  Library name . . . . . :     UDCS_APPLS
Source file . . . . . :   QCSRC
  Library name . . . . . :     UDCS_APPLS
Source member name . . . . . :   TARGET
Text Description . . . . . :  Target Application Example
Compiler options . . . . . :   *SOURCE  *NOXREF *NOSHOWUSR
                           :   *NOSHOWSYS *NOSHOWSKP *NOEXPMAC
                           :   *NOAGR *NOPPONLY *NODEBUG
                           :   *GEN  *NOSECLVL  *PRINT  *LOGMSG
Language level options . . . . :
Source margins:
  Left margin . . . . . :   1
  Right margin . . . . . :   80
Sequence columns:
  Left Column . . . . . :
  Right Column . . . . . :
Define name . . . . . :
Generation options . . . . . :   *NOLIST  *NOXREF  *GEN *NOATR
                           :   *NODUMP  *NOOPTIMIZE  *NOALWBND
                           :   *NOANNO
Print file . . . . . :   QSYSPRT
  Library name . . . . . :     *LIBL
Message flagging level . . . . :   0
Compiler message:
  Message limit . . . . . :   *NOMAX
  Message limit severity . . . :   30
Replace program object . . . . :   *YES
User profile . . . . . :   *USER
Authority . . . . . :   *LIBCRTAUT
Target Release . . . . . :   *CURRENT
INDEBUG options . . . . . :   I don't know
Last change . . . . . :   90/12/19 08:49:37
Source description . . . . . :   Target Application Example
Compiler . . . . . :   IBM SAA C/400 Compiler

/*****************************************************************/
/**                                                             **/
/**  Program Name: Target Application Program Example           **/
/**                                                             **/
/**                                                             **/
/**  Function:                                                  **/
/**  This is the target application program example that uses   **/
/**  X.25 services provided by the user-defined communications  **/
/**  support to receive a simple file from the source application **/
/**  program running on System A.  This program performs the    **/
/**  following:                                                 **/
/**     01.  Open the target file named OUTFILE.                **/
/**     02.  Call QOLQLIND to obtain local line information.    **/
/**     03.  Enable a link.                                     **/
/**     04.  Set a Filter on the enabled link.                  **/
/**     05.  Receive a 'B101'X operation (incoming call).       **/
/**     06.  Send a 'B400'X operation (accept call).            **/
/**     07.  Receive '0001'X operation(s) (incoming data) from  **/
/**          the source application program and write it to the **/
/**          file opened in step 1).                            **/
/**     08.  Receive a 'B301'X operation (clear call indication). **/
/**     09.  Send a 'B100'X operation to respond locally to the **/
/**          clearing of the connection.                        **/
/**     10.  Disable the link enabled in step 3).               **/
/**                                                             **/
/**  A data queue will be actively used to manage the operation **/
/**  of this program.  Data queue support will be used to monitor **/
/**  for the completion of the enable and disable routines, as  **/
/**  well as timer expirations and incoming data.  Timers are   **/
```

Figure A-2 (Part 1 of 15). C/400 Compiler Listing for the Target Application

```
/**  used to ensure that there will never be an infinite wait on  **/
/**  the data queue.  If a timer expires, the link enabled will   **/
/**  be disabled and the program will stop.                       **/
/**                                                               **/
/**                                                               **/
/** Inputs:                                                       **/
/** The program expects the following input parameters:           **/
/**      Line Name:   This is the name of the line description    **/
/**                   that will be used to call the QOLELINK API. **/
/**                   The line must be an X.25 line with at least **/
/**                   one SVC of type *SVCBOTH or *SVCIN.         **/
/**                                                               **/
/**      CommHandle:  This is the logical name that will be used  **/
/**                   to identify the link enabled.               **/
/**                                                               **/
/**      Remote DTE Address:  The is the Local Network Address    **/
/**                           of system A.                        **/
/**                                                               **/
/**                                                               **/
/** Outputs:                                                      **/
/** Current status of the file transfer will be provided when     **/
/** running this program. If an error should occur, then a        **/
/** message will be displayed indicating where the error occurred **/
/** and the program will end.        If the program completes     **/
/** successfully, a "successful completion" message will be       **/
/** posted.                                                       **/
/**                                                               **/
/*******************************************************************/

#include "header"

void senddata(sendparms *a, char *b, desc *c, char *d, char *e, int f);

void sndformat1(sendparms *a,char *b, char *c, char *d, qlindparms *e);

void sndformat2 (sendparms *a, char *b, char *c);

void setfilters (hdrparms *a);

void byte (char *a, int b, char *c, int d);

void printespec (espec *a);

void settimer(unsigned short *a,char *b,qentry *c,usrspace *d,char *e);

void dequeue (int a, char *b, qentry *c, usrspace *d);

void putdata (char *a, int b, FILE *c);

void x25lind (qlindparms *a, char *b);

void disablelink (disableparms *a, char *b, usrspace *c);

void handler (disableparms a, usrspace *b);

sigdata_t *sigdata(void);

/*********************************************************/
/***************   Start Main Program   ******************/
/*********************************************************/

main (int argc, char *argv??(??))
{
```

Figure A-2 (Part 2 of 15). C/400 Compiler Listing for the Target Application

```
/************  Variable  Declarations      ******************/

  usrspace inbuff,       /*  Input Data Buffer */
           indesc,       /*  Input Buffer Descriptor  */
           outbuff,      /*  Output Data Buffer  */
           outdesc,      /*  Output Buffer Descriptor */
           qname;        /*  Data Queue  */

  int length,                /* Data Queue key length  */
      inc, i, j;                    /*  counters  */
  unsigned short expctid;      /*  Message ID that is expected */
  char commhandle??(10??),     /*  Command Line Parameter  */
       rmtdte??(17??),         /*  Remote DTE Address  */
       *buffer,                /* Pointer to buffer     */
       key??(256??);           /* Data Queue key identifier  */
  desc *descriptor;            /* Pointer to buffer descriptor  */
/**  definitions for API functions  **/
  enableparms enable;
  disableparms disable;
  sendparms send;
  recvparms recv;
  setfparms setf;
  timerparms timer;
  qlindparms qlind;
  qentry dataq;
  hdrparms *header;

  /******  Annnddddddd....  there off!!   ***********/
1

  /***---  Open the file to put the received data.     ----**/
  if ((fptr = fopen("UDCS_APPLS/OUTFILE)", "w")) == NULL)
    {
     printf("Unable to open target output file in UDCS_APPLS LIB.\n");
     printf("The Program was terminated.\n\n");
     return;
    }
  /***---  Open the display file for error handling.   ----**/
  if ((screen = fopen("ERRORSPEC", "ab+ type = record")) == NULL)
    {
     printf("Unable to open display file.\n");
     printf("The Program was terminated.\n\n");
     return;
    }

  /***---  Set the Exception Handler        ----**/
  signal(SIGABRT,&handler);

  /**  Clear the command line parameters  **/
  strncpy(enable.linename, "          ", 10);    /* Clear linename */
  strncpy(commhandle, "          ", 10);     /* Clear Commhandle */
  strncpy(rmtdte, "                ", 17);     /* Clear Remote DTE */

  /**  Receive command line Parameters  **/
  strncpy(enable.linename, argv??(1??), strlen(argv??(1??)));
  strncpy(commhandle, argv??(2??), strlen(argv??(2??)));
  strncpy(rmtdte, argv??(3??), strlen(argv??(3??)));
  rmtdte??(strlen(argv??(3??))??) = '\0';

  /**   Initialize the user spaces  **/
  strncpy(inbuff.library, "UDCS_APPLS", 10);    /*  Input Buffer */
  strncpy(inbuff.name, "TARGETIBUF", 10);
  strncpy(indesc.library, "UDCS_APPLS", 10);    /* Input B Desc  */
  strncpy(indesc.name, "TARGETIDSC", 10);
```

Figure A-2 (Part 3 of 15). C/400 Compiler Listing for the Target Application

```
strncpy(outbuff.library, "UDCS_APPLS", 10);   /*  Output Buffer*/
strncpy(outbuff.name, "TARGETOBUF", 10);
strncpy(outdesc.library, "UDCS_APPLS", 10);   /* Output B Desc */
strncpy(outdesc.name, "TARGETODSC", 10);
strncpy(qname.library, "UDCS_APPLS", 10);   /*  Data queue  */
strncpy(qname.name, "X25DTAQ   ", 10);

/*****   retrieve the line description information  ******/
x25lind (&qlind, enable.linename);
if ((qlind.retcode != 0) || (qlind.reason != 0))
   {
   printf("Query line description failed.\n");
   printf("Return code = %d\n", qlind.retcode);
   printf("Reason code = %d\n\n", qlind.reason);
   return;
   }

/*****   Hard Code the QOLELINK Input Parameters  ******/
enable.maxdtax25 = 512;
enable.keylength = 3;
strncpy(enable.keyvalue, "RCV", 3);
```

2

```
/**-------  Enable the link  -----------**/
QOLELINK (&(enable.retcode), &(enable.reason), &(enable.tdusize),\
    &(enable.numunits), &(enable.maxdtalan), &(enable.maxdtax25),\
    (char *)&inbuff, (char *)&indesc, (char *)&outbuff,\
    (char *)&outdesc, &(enable.keylength), enable.keyvalue,\
    (char *)&qname, enable.linename, commhandle);

if ((enable.retcode != 0) || (enable.reason != 0))
   {
   printf("Line %.10s with Commhandle %.10s was NOT ENABLED.\n",\
           enable.linename, commhandle);
   printf("Return code = %d\n", enable.retcode);
   printf("Reason code = %d\n\n", enable.reason);
   return;
   }
```

3

```
/**-------  Set a timer for Enable link  --------**/
expctid = 0xF0F0;
settimer(&expctid, "Enable", &dataq, &qname, commhandle);
if (expctid != 0xF0F0)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }
```

Figure A-2 (Part 4 of 15). C/400 Compiler Listing for the Target Application

```
/*************************************************************/
/*******----  Set a Filter for the Link  --------*********/
/*************************************************************/
```

4

```
QUSPTRUS(&outbuff, &header);   /*  get the output buffer pointer */
header->function = 1;          /*  add a filter   */
header->type = 0;              /*  X.25 PID only  */
header->number = 1;            /*  set 1 filter   */
header->length = 16;           /*  X.25 filter length  */
setfilters(header);            /*  Fill in the filter format */

/*******----  Set the filter for the Link  --------*********/
QOLSETF (&(setf.retcode), &(setf.reason), &(setf.erroffset),\
            commhandle);
if ((setf.retcode != 0) || (setf.reason != 0))
   {
   printf("Set Filters Return Code = %.2d\n", setf.retcode);
   printf("Set Filters Reason Codes = %.4d\n", setf.reason);
   printf("Set Filters Error Offset = %.4d\n", setf.erroffset);
   return;
   }

/*************************************************************/
/****  Receive the incoming call packet and accept the call **/
/*************************************************************/

/**-------  Set a timer to receive data  --------**/
expctid = 0xF0F3;
settimer(&expctid, "Inc Call ", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }
```

5

```
/**********  Receive the Incoming Data     **********/
QUSPTRUS (&inbuff, &buffer);
QUSPTRUS (&indesc, &descriptor);

QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
         &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
         &(recv.dataavail), &(recv.errorspecific), commhandle);

if ((recv.retcode != 0) || (recv.reason != 0))
   {
   printf("Recv incoming call packet failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

/*** Interpret the Received Operation  ***/
if (recv.operation != 0xB201)
   {
   printf("Recvd operation %x instead of B201", recv.operation);
   disablelink (&disable, commhandle, &qname);
   return;         /****  End the program  ***/
   }
```

Figure A-2 (Part 5 of 15). C/400 Compiler Listing for the Target Application

6

```
/*****************************************************************/
/** Send a response to accept the call and establish a connection */
/*****************************************************************/

/****   Get pointers to the user spaces.   ******/
QUSPTRUS(&outbuff, &buffer);
QUSPTRUS(&outdesc, &descriptor);

/*******   Set up Send Packet    *********/
send.ucep = 62;                    /*  set UCEP to be 62  */
send.pcep = recv.pcep;             /*  get the PCEP number  */
send.operation = 0xB400;           /*  send a call request response*/
send.numdtaelmnts = 1;             /*  send one data unit   */

/**-----   Send the packet   ----------------**/
sndformat1 (&send, buffer, rmtdte, commhandle, &qlind);
if ((send.retcode != 0) || (send.reason != 0))
   {
   printf("Data NOT sent for commhandle %.9s\n", commhandle);
   printf("Return code = %d\n", send.retcode);
   printf("Reason code = %d\n", send.reason);
   printf("new pcep %d\n\n", send.newpcep);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

printf("An X.25 SVC connection was completed\n\n");
```

7

```
/*********************************************************/
/****    Receive Incoming Data    ************************/
/*********************************************************/

/**-------  Set a timer to receive data  --------**/
expctid = 0xF0F3;
settimer(&expctid, "Inc Data ", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }

/*******---  Receive the Incoming Data     ----******/
/** Get pointer to user space  **/
QUSPTRUS (&inbuff, &buffer);
QUSPTRUS (&indesc, &descriptor);

/**  Receive the data   **/
QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
       &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
       &(recv.dataavail), &(recv.errorspecific), commhandle);

if ((recv.retcode != 0) || (recv.reason != 0))
   {
   printf("Recv op for first data unit failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));
```

Figure A-2 (Part 6 of 15). C/400 Compiler Listing for the Target Application

```
   disablelink (&disable, commhandle, &qname);
   return;
   }
```

8

```
/*************************************************************/
/*******   Start a loop to read in all the incoming data    ***/
/*************************************************************/

i = 1;
while (recv.operation == 0x0001)
   {
   printf("%d Data Recvd {%.4x}.\n\n", i++, recv.operation);

   /**  Store all the data units in the file  **/
   for (j = 1; j <= recv.numdtaunits; j++) {
      putdata (buffer + (j - 1)*enable.tdusize,\
               descriptor->length, fptr);
      descriptor = (desc *)((char *)descriptor + sizeof(desc));
      }   /* for */

   /**-------  Set a timer to wait for more data  -------**/
   if (recv.dataavail == 0)
      {
      /** Set timer  **/
      expctid = 0xF0F3;
      settimer(&expctid, "Wt Inc Dta", &dataq, &qname, commhandle);
      if (expctid != 0xF0F3)
         {
         disablelink (&disable, commhandle, &qname);
         return;
         }
      }

   /** Get pointer to user space  **/
   QUSPTRUS (&inbuff, &buffer);
   QUSPTRUS (&indesc, &descriptor);

   /**  Receive the data   **/
   QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
          &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
          &(recv.dataavail), &(recv.errorspecific), commhandle);

   }    /**  End  Receive data  while loop   ******/
```

9

```
/*************************************************/
/*********** Receive the Clear indication ***********/
/*************************************************/

if ((recv.retcode != 83) || (recv.reason != 4002))
   {
   printf("Recv opr for clear request failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }
```

Figure A-2 (Part 7 of 15). C/400 Compiler Listing for the Target Application

```
/* Interpret the Received Operation  */
if (recv.operation != 0xB301)
   {
   printf("Recvd operation %x instead of B301", recv.operation);
   disablelink (&disable, commhandle, &qname);
   return;         /****  end the program  ***/
   }
```

10

```
/*************************************************************/
/***********  Send local response to clear indication ***********/
/*************************************************************/
/****   Get pointers to the user spaces.   ******/
QUSPTRUS(&outbuff, &buffer);
QUSPTRUS(&outdesc, &descriptor);

/*******   Set up the packet         ****************/
send.operation = 0xB100;       /*  send a clear request packet  */
send.numdtaelmnts = 1;         /*  send one data unit  */

/**-----   Send the packet   ----------------**/
sndformat2 (&send, buffer, commhandle);

if ((send.retcode != 0) && (send.reason != 0))
   {
   printf("Response not sent for clear connection\n");
   printf("Return code = %d\n", send.retcode);
   printf("Reason code = %d\n", send.reason);
   printf("new pcep %d\n\n", send.newpcep);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }


/*****************************************************/
/*********** Receive the Clear Confirmation  **********/
/*****************************************************/

/**-------  Set a timer to receive data  --------**/
expctid = 0xF0F3;
settimer(&expctid, "Clr Cnfrm", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }

if ((recv.retcode != 00) || (recv.reason != 0000))
   {
   printf("Recv failed for clear confirmation\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }
```

Figure A-2 (Part 8 of 15). C/400 Compiler Listing for the Target Application

```
  /* Interpret the Received Operation  */
  if (recv.operation != 0xB101)
     {
     printf("Recvd opr %x instead of opr B301\n", recv.operation);
     disablelink (&disable, commhandle, &qname);
     return;
     }
11
  /***************************************/
  /**  disable the link and end program  **/
  /***************************************/

  disablelink (&disable, commhandle, &qname);

  printf("TARGET application completed OK!\n\n");

}    /* End Main */

/*********************************************************************/
/************** Start Subroutine Section    **********************/
/*********************************************************************/

/*********************************************************************/
/************  Routine to fill X.25 Format I   ***********/

void sndformat1 (sendparms *send,
                 char *buffer,
                 char *rmtdte,
                 char *commhandle,
                 qlindparms *qlind)

{
  format1 *output = (format1 *) buffer;
  register int counter;
  register querydata *qd;

  qd = (querydata *)&(qlind->userbuffer);
  output->type = 0;                  /*  not used  */
  output->logchanid = 0x0;
  output->sendpacksize = qd->x25data.defsend;
  output->sendwindsize = qd->x25data.windowsend;
  output->recvpacksize = qd->x25data.defrecv;
  output->recvwindsize = qd->x25data.windowrecv;

  output->dtelength = strlen(rmtdte);              /*  not used  */
  byte(output->dte, 16, rmtdte, strlen(rmtdte));   /*  not used  */
  output->dbit = 0;
  output->cug = 0;                                 /*  not used  */
  output->cugid = 0;                               /*  not used  */
  output->reverse = 0;                             /*  not used  */
  output->fast = 0;                                /*  not used  */
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->calllength = 0;
  byte(output->callud, 128, "00", 2);
  output->misc??(0??) = 0;
  output->misc??(1??) = 0;
  output->misc??(2??) = 0;
  output->misc??(3??) = 0;
  output->maxasmsize = 16383;
  output->autoflow = 32;
```

Figure A-2 (Part 9 of 15). C/400 Compiler Listing for the Target Application

```
  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));

} /*  End sndformat1 Subroutine  */
/*****************************************************************/
/************  Routine to fill X.25 Format II   ************/

void sndformat2 (sendparms *send,
                 char *buffer,
                 char *commhandle)

{
  format2 *output = (format2 *) buffer;

  output->type = 1;
  output->cause = 'FF';
  output->diagnostic = 'FF';
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->length = 0;
  byte(output->userdata, 128, "", 0);


  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));

} /*  End sndformat2 Subroutine  */

/*****************************************************************/
/**************   Fill in the Buffer for the Filter  ****************/

void setfilters (hdrparms *header)

{
  x25filter *filters;

  filters = (x25filter *)header->filters;
  filters??(0??).pidlength = 1;
  filters??(0??).pid = 0x21;            /*  set the protocol ID  */
  filters??(0??).dtelength = 0;         /* no DTE used in filter */
  byte(filters??(0??).dte, 12, "", 0);
  filters??(0??).flags = 0x0;
  filters??(0??).flags += 0x80;      /* Set Reverse Charging to no  */
  filters??(0??).flags += 0x40;      /*  Set Fast Select to no  */

} /*  End setfilters Subroutine  */

/*****************************************/
/********    Routine to disable   ************/

void disablelink (disableparms *disable,
                  char *commhandle,
                  usrspace *qname)

{
  qentry dataq;
  unsigned short expctid;
```

Figure A-2 (Part 10 of 15). C/400 Compiler Listing for the Target Application

```
  disable->vary = 1;     /*  Hard code device to vary off  */

  /**  Call disable link  **/
  QOLDLINK (&(disable->retcode), &(disable->reason),\
                       commhandle, &(disable->vary));

  if ((disable->retcode != 0) && (disable->reason != 00))
     {
     printf ("Link %.10s did not disabled.\n", commhandle);
     printf ("return code = %d\n", disable->retcode);
     printf ("reason code = %d\n\n", disable->reason);
     }
  else
     printf ("%.10s link disabled \n", commhandle);

  /**-------  Set a timer to receive message  --------**/
  expctid = 0xF0F1;
  settimer(&expctid, "Disable  ", &dataq, qname, commhandle);
  if (expctid != 0xF0F1)
     {
     printf("Disable link did not complete successfully");
     return;
     }

  /**  close the files  **/
  fclose(fptr);
  fclose(screen);

}  /*  End disablelink Subroutine  */


/*****************************************************************/
/**    Routine to convert string to Hexadecimal format    ******/

void byte (char *dest,
           int dlength,
           char *source,
           int slength)

{
  register int counter;
  char holder??(2??);

  for (counter=0;counter<dlength;counter++)
   dest??(counter??)=0;
  for (counter=slength-1;counter>=0;counter--)
   if isxdigit(source??(counter??))
     {
      holder??(0??)=source??(counter??);
      holder??(1??)='\0';
      if  (counter % 2 == 0)
        dest??(counter/2??) += (char) hextoint(holder)*16;
      else dest??(counter/2??) += (char) hextoint(holder);
     }

}  /*  End byte Subroutine  */

/*****************************************************************/
/**    Routine to display the ErrorSpecific output       ******/
/*****************************************************************/

void printespec(espec *errorspecific)
```

Figure A-2 (Part 11 of 15). CI400 Compiler Listing for the Target Application

```
{
  especout outparms;

  QXXFORMAT(screen, "ERRORSPEC ");
  sprintf(outparms.hwecode, "%.8X", errorspecific->hwecode);
  sprintf(outparms.timestamp, "%.8X%.8X", errorspecific->timestamphi,\
                                          errorspecific->timestamplo);
  sprintf(outparms.elogid, "%.8X", errorspecific->elogid);
  if (errorspecific->flags & 0x40)
    outparms.fail = 'Y';
  else outparms.fail = 'N';
  if (errorspecific->flags & 0x20)
    outparms.zerocodes = 'Y';
  else outparms.zerocodes = 'N';
  if (errorspecific->flags & 0x10)
    outparms.qsysopr = 'Y';
  else outparms.qsysopr = 'N';
  sprintf(outparms.cause,"%.2X", errorspecific->cause);
  sprintf(outparms.diagnostic, "%.2X", errorspecific->diagnostic);
  sprintf(outparms.erroffset, "%.6d", errorspecific->erroroffset);
  fwrite(&outparms, 1, sizeof(especout), screen);
  fread("", 0, 0, screen);

}  /*  End printespec Subroutine  */


/*****************************************************************/
/*******   Dequeues the Incoming Message and processes it ******/

void dequeue (int length,
              char *key,
              qentry *dataq,
              usrspace *qname)

{
  char fldlen??(3??),
       waittime??(3??),
       keylen??(2??),
       senderid??(2??),
       *pointer,
       order??(2??);
  register int counter;

  waittime??(0??) = 0;
  waittime??(1??) = 0;
  waittime??(2??) = 0x1D;   /*  Hard code a delay of infinite   */
  keylen??(0??) = 0;
  keylen??(1??) = 0x6F;  /* Hard code a keylength of 6  */
  senderid??(0??) = 0;
  senderid??(1??) = 0x0F;
  strncpy(order, "EQ", 2);

  /*  Clear the data structures  **/
  fflush(stdin);
  pointer = (char *)dataq;
  for (counter = 0; counter < 336; counter++)
    pointer??(counter??) = 0;

  strncpy (dataq->type, "       ", 7);
  while ((strncmp(dataq->type, "*USRDFN", 7) != 0) || (fldlen == 0))
     QRCVDTAQ(qname->name, qname->library, fldlen, dataq, waittime,\
           order, keylen, key, senderid,"");

}  /*  End dequeue Subroutine  */
```

Figure A-2 (Part 12 of 15). C/400 Compiler Listing for the Target Application

```
/*********************************************************/
/*************   Set a timer and dequeue next entry  ******/

void settimer (unsigned short *expctid,
               char *process,
               qentry *dataq,
               usrspace *qname,
               char *commhandle)

{
timerparms timer;
disableparms disable;
int length;
char key??(6??);

  timer.interval = 20000;             /*  set timer for 20 seconds */
  timer.establishcount = 1;            /*  set establish count to 1 */
  timer.keylength = 6;                 /*  key value  */
  strncpy(timer.keyvalue, "TARGET", 6);   /*  set key value /
  timer.operation = 1;                 /*  set a timer  */

  /* Call QOLTIMER   */
  QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
           timer.handlein, (char *)qname, &(timer.operation),\
           &(timer.interval), &(timer.establishcount),\
           &(timer.keylength), timer.keyvalue, timer.userdata);

  if ((timer.retcode != 0) || (timer.reason != 0))
     {
     printf("%s timer failed while being set.\n", process);
     printf("Return code = %d\n", timer.retcode);
     printf("Reason code = %d\n\n", timer.reason);
     }

  /**-------  Dequeue an entry  --------**/
  strncpy(key, "TARGET", 6);
  length = 6;
  dequeue (length, key, dataq, qname);

  /***----     Cancel timer   -----***/
  if (dataq->msgid != 0xF0F4)
     {
     strncpy(timer.handlein, timer.handleout, 8);
     timer.operation = 2;           /*  Cancel one timer  */

     QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
           timer.handlein, (char *)qname, &(timer.operation),\
           &(timer.interval), &(timer.establishcount),\
           &(timer.keylength), timer.keyvalue, timer.userdata);

     if ((timer.retcode != 0) || (timer.reason != 0))
        {
        printf("%s timer failed while being canceled\n", process);
        printf("Return code = %d\n", timer.retcode);
        printf("Reason code = %d\n\n", timer.reason);
        }
     }
  if (dataq->msgid != *expctid)
     {
     printf ("A %.4X message ID was received instead of %.4X\n",\
              dataq->msgid, *expctid);
```

Figure A-2 (Part 13 of 15). C/400 Compiler Listing for the Target Application

```
     printf ("%s completion message was not received\n", process);
     *expctid = dataq->msgid;
     }

} /* End settimer Subroutine */

/*************************************************************/
/** x25lind: Read a record into buf and return length   **/

void x25lind (qlindparms *qlind, char *linename)
{
register int counter;

  for(counter=0;counter<256;counter++)
     qlind->userbuffer??(counter??)=0;

  qlind->format = 0x01;
  QOLQLIND (&(qlind->retcode), &(qlind->reason), &(qlind->nbytes),\
           qlind->userbuffer, linename, &(qlind->format));

} /* End x25lind Subroutine */

/*************************************************************/
/** putdata: Read a record into buf and return length   **/

void putdata (char *buf,
              int dtalen,
              FILE *fptr)

{
int i;

   for (i = 0; i < dtalen; i++)
      fwrite(buf + i, 1, 1, fptr);

} /* End putdata Subroutine */


/* Exception handler, so that a failure will not
    kill the program, and any associated data!! */

void handler(disableparms disable, usrspace *qname)
{
  sigdata_t *data;

  disablelink(&disable, "*ALL      ", qname);
  printf("The program received an excecption.\n");
  printf("Disable Link was called & the program was terminated.\n\n");

  data=sigdata();
  data->sigact->xhalt=0;
  data->sigact->xrtntosgnler=0;
  data->sigact->xresigprior=0;
  data->sigact->xresigouter=0;

} /* End handler Subroutine */

     * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure A-2 (Part 14 of 15). C/400 Compiler Listing for the Target Application

```
    * * * * *  I N C L U D E S  * * * * *
 Include Name     Last change        Actual Include Name
 header           90/12/19 08:49:31  UDCS_APPLS/QCSRC/HEADER
 typedefs         90/12/19 08:49:31  UDCS_APPLS/QCSRC/TYPEDEFS
 stdio.h          90/11/12 17:24:55  QCC/H/STDIO
 stddef.h         90/11/12 17:24:54  QCC/H/STDDEF
 errno.h          90/11/12 17:24:49  QCC/H/ERRNO
 signal.h         90/11/12 17:24:53  QCC/H/SIGNAL
 ctype.h          90/11/12 17:24:49  QCC/H/CTYPE
 stdarg.h         90/11/12 17:24:54  QCC/H/STDARG
 stdlib.h         90/11/12 17:24:56  QCC/H/STDLIB
 signal.h         90/11/12 17:24:53  QCC/H/SIGNAL
 xxasio.h         90/11/12 17:24:57  QCC/H/XXASIO
 xxcvt.h          90/11/12 17:24:58  QCC/H/XXCVT
 string.h         90/11/12 17:24:56  QCC/H/STRING
 ctype.h          90/11/12 17:24:49  QCC/H/CTYPE
 hexconv          90/12/19 08:49:27  UDCS_APPLS/QCSRC/HEXCONV
 stdio.h          90/11/12 17:24:55  QCC/H/STDIO
 * * * * *  E N D  O F  I N C L U D E S  * * * * *


* * * * *  M E S S A G E  S U M M A R Y  * * * * *
Total  Info   Warning   Error    Severe    Terminal
       (0-4)  (5-19)  (20-29)   (30-39)    (40-99)
  0      0       0        0         0          0
* *  E N D  O F  M E S S A G E  S U M M A R Y  * *


IBM SAA C/400           UDCS_APPLS/TARGET
 ROUTINE          BLOCK NUMBER  SCOPE  TYPE
 <MAIN>                 2       LOCAL  MAIN-PROGRAM
 __sigdata              6       LOCAL  PROCEDURE
 __sgnl                12       LOCAL  PROCEDURE
 __frdinit             49       LOCAL  PROCEDURE
 __getrec              50       LOCAL  PROCEDURE
 __putrec              51       LOCAL  PROCEDURE
 __frdexit             52       LOCAL  PROCEDURE
 __fwrinit             53       LOCAL  PROCEDURE
 __fwrexit             54       LOCAL  PROCEDURE
 QXXFORMAT            129       LOCAL  PROCEDURE
 __strlen             164       LOCAL  PROCEDURE
 __strncmp            168       LOCAL  PROCEDURE
 __strncpy            170       LOCAL  PROCEDURE
 inttohex             181       ENTRY   PROCEDURE
 hextoint             182       ENTRY   PROCEDURE
 sndformat1           197       ENTRY   PROCEDURE
 sndformat2           198       ENTRY   PROCEDURE
 setfilters           199       ENTRY   PROCEDURE
 byte                 200       ENTRY   PROCEDURE
 printespec           201       ENTRY   PROCEDURE
 settimer             202       ENTRY   PROCEDURE
 dequeue              203       ENTRY   PROCEDURE
 putdata              204       ENTRY   PROCEDURE
 x25lind              205       ENTRY   PROCEDURE
 disablelink          206       ENTRY   PROCEDURE
 handler              207       ENTRY   PROCEDURE
 main                 208       ENTRY   PROCEDURE
Program TARGET was created in library UDCS_APPLS.
* * *  E N D  O F  C O M P I L A T I O N  * * *
```

Figure A-2 (Part 15 of 15). C/400 Compiler Listing for the Target Application

The following reference numbers and explanations correspond to the reference numbers in the target application's program listing.

1 Call the C library routines fopen() and signal() to open the target file and set up a signal handler to process AS/400 exceptions, respectively. If an exception situation is encountered, the handler() will

be called to perform clean-up in order for the program to end.

2 Call the QOLELINK API to enable the line description using the line name and communications handle passed as input parameters to this program.

3 Call the QOLTIMER API to time the completion of the enable link operation. If the timer expires before the enable-complete message is posted on the this program's data queue. then this program will end.

4 Call the QUSPTRUS API to obtain a pointer to the beginning of the output buffer user space. The output buffer will be used to construct a filter list for the call to the QOLSETF API.

5 Call the QOLRECV API to receive inbound data after reading an incoming data message that was posted on the program's data queue by the user-defined communications support. Since these programs are operating using the communications services of X.25, the first data unit the target program should see is a X'B201' operation signalling an incoming call was received.

6 Call the QOLSEND API with a X'B400' operation to accept the incoming X.25 call. A connection is now established between the source and target application programs.

7 The target program will now set a timer by calling the QOLTIMER API and wait for incoming data. If the timer expires before any incoming data is received, then this program will call the QOLDLINK API, and end.

8 This is the main receive loop for the target program. When data is received from the source program, it will be written to the target file opened during the initialization of this program. The loop will process until a message other than incoming-data entry is read from the program's data queue.

9 Call the QOLSEND API with a X'B001' operation to locally close the connection.

10 Receives a X'B101' operation from the user-defined communications support. This is a local confirmation of X'B100' operation.

11 Call the QOLDLINK API to disable the link previously enabled and end.

Bibliography

This bibliography lists printed information that you need to use the OS/400 APIs, background information for the functions the APIs perform, and other information relevant to specific types of applications. The manuals are grouped in these categories:

- General-purpose manuals
- Programming language manuals
- Communications manuals
- Document library services (DLS) file system manuals

The lists below give the full title and order number of each manual. When these manuals are referred to in text, a shortened version of the title is used.

If you want more information on a topic while you are using this reference, see the *Publications Guide*, GC41-9678, for related AS/400 publications.

General-Purpose Manuals

These books provide general-purpose and background information for the OS/400 licensed program:

- *Advanced Backup and Recovery Guide*, SC41-8079

  The *Advanced Backup and Recovery Guide* provides information about planning a backup and recovery strategy.

- *Basic Security Guide*, SC41-0047

  The *Basic Security Guide* provides general information about OS/400 security and authorities.

- *Central Site Distribution Guide*, SC41-9993

  The *Central Site Distribution Guide* provides information on how to distribute licensed programs, program temporary fixes (PTFs), and application programs to other systems under IBM licensing agreements.

- *Data Description Specifications Reference*, SC41-9620

  The *DDS Reference* provides detailed descriptions of the entries and keywords needed to describe database files (both logical and physical) and certain device files (for displays, printers, and ICF) external to the user's programs.

- *Database Guide*, SC41-9659

  The *Database Guide* describes database concepts.

- *Device Configuration Guide*, SC41-8106

  The *Device Configuration Guide* provides information on how to configure hardware initially and how to change that configuration. Also included is a description of the different keyboard language types. Keyboard language types are specified when using the TELNET function.

- *Guide to Programming Application and Help Displays*, SC41-0011

  The *Guide to Programming Displays* provides information about:

  - Creating and working with display files
  - Developing online information
  - Creating panel group objects

  This manual also describes the method by which AS/400 users can send a 5250 data stream to a display device (user-defined data streams).

- *National Language Support Planning Guide*, GC41-9877

  The *National Language Support Planning Guide* provides information needed to evaluate, plan, and use the AS/400 national language support (NLS) and multilingual capabilities.

- *New User's Guide*, SC41-8211

  The *New User's Guide* provides information about how to sign on and off, send and receive messages, respond to keyboard error messages, use function keys, and control and manage jobs. Also included is a description of keyboard differences.

- *Programming: Control Language Programmer's Guide*, SC41-8077

  The *CL Programmer's Guide* discusses OS/400 functions and concepts that are relevant to programming. It includes descriptions of special-purpose APIs not covered in this manual, such as APIs for command syntax checking and data queue management.

- *Programming: Control Language Reference*, SC41-0030

  The *CL Reference* manual provides a description of the AS/400 control language (CL) and its commands. Each command description includes a syntax diagram, parameters, default values, keywords, and an example.

- *Programming: Reference Summary*, SX41-0028

  The *Programming Reference Summary* provides quick reference information when working with the AS/400 system. This manual contains summaries of information such as system values and OS/400 data description specifications (DDS) keywords.

  If you want to create your own translation tables, this manual discusses creating and editing tables for ASCII line mode.

- *Programming: Work Management Guide*, SC41-8078

  The *Work Management Guide* describes work and job management concepts.

- *Security Reference*, SC41-8083

  The *Security Reference* provides technical information about OS/400 security.

- *Service: Diagnostic Aids — Volume 1*, LY44-0597

  The *Diagnostic Aids — Volume 1* provides a list of available object types in hexadecimal format for use with the object APIs.

- *System Concepts*, GC41-9802

The *System Concepts* provides a general understanding of the concepts related to the overall design and use of the AS/400 system and its operating system. This manual includes general information about AS/400 functions such as user interface, object management, work management, system management, data management, database, communications, environments, OfficeVision/400, PC Support, and system architecture.

- *System Operator's Guide*, SC41-8082

  The *Operator's Guide* provides information on how to respond to error messages and process and manage jobs on the system. Processing jobs includes working with spooled files and finding your printer output.

Programming Language Manuals

You might refer to these programming language manuals while writing applications with the OS/400 APIs:

- *Languages: Pascal Reference*, SC09-1210

  The *Pascal Reference* provides information about using the AS/400 Pascal programming language. It includes information you can refer to while using AS/400 Pascal, such as a detailed description of the program structure, declarations, data types, routines, variables, and statements in Pascal.

- *Languages: Pascal User's Guide*, SC09-1209

  The *Pascal User's Guide* explains how to enter, compile, run, and debug AS/400 Pascal programs.

- *Languages: Systems Application Architecture* AD/Cycle* COBOL/400* User's Guide*, SC09-1383

  The *COBOL/400* User's Guide* provides information needed to design, write, test, and maintain COBOL/400 programs on the AS/400 system.

- *Languages: Systems Application Architecture* AD/Cycle* RPG/400* Reference*, SC09-1349

  The *RPG/400* Reference* provides information needed to write programs for the AS/400 system using the RPG/400 programming language. This manual describes, position by position, the valid entries for all RPG specification forms, and provides a detailed description of all the operation codes. This manual also contains information on the RPG logic cycle, arrays and tables, editing functions, and indicators.

- *Languages: Systems Application Architecture* AD/Cycle* RPG/400* User's Guide*, SC09-1348

  The *RPG/400* User's Guide* provides information needed to write, test, and maintain RPG/400 programs on the AS/400 system. The manual provides information on data organizations, data formats, file processing, multiple file processing, automatic report function, RPG command statements, testing and debugging functions, application design techniques, problem analysis, and compiler service information. The differences between the System/38 RPG III, System/38 compatible RPG, and RPG/400 are identified.

- *Languages: Systems Application Architecture* C/400* User's Guide*, SC09-1347

  The *C/400* User's Guide* provides information needed to write application programs or develop programs using the C/400 language.

- *Languages: System C/400 Programming RPQ P10102 User's Guide and Reference*, SC09-1317

  This manual describes how to enter, compile, test, and debug System C/400 PRPQ programs.

- *Machine Interface Functional Reference*, SC41-8226

  The *MI Functional Reference* is a comprehensive reference to machine interface (MI) instructions.

- *Programming: GDDM Programming Guide*, SC41-0536

  The *GDDM Programming Guide* provides information about using OS/400 graphical data display manager (GDDM) to write graphics application programs.

- *Programming: Procedures Language 400/REXX Programmer's Guide*, SC24-5553,

  The *Procedures Language (REXX) Programmer's Guide* explains Procedures Language 400/REXX programming concepts and discusses considerations in using this language on the AS/400 system. It also describes REXX APIs and provides examples that you can use to learn Procedures Language 400/REXX.

- *Programming: Procedures Language 400/REXX Reference*, SC24-5552

  The *Procedures Language (REXX) Reference* provides an overview of the Procedures Language 400/REXX concepts and includes information about keyword instruction syntax, function syntax, numerics, arithmetic, conditions, input and output streams, testing, and double-byte character set (DBCS) support. The manual also describes REXX APIs.

- *SAA* AD/Cycle* Application Development Manager/400 Application Developer's Guide*, SC09-1375

  The *Application Development Manager/400 Application Developer's Guide* provides information needed for Application Development Manager/400 developers to perform tasks on the components of an application in a project hierarchy that is defined by the project administrator.

- *SAA* AD/Cycle* Application Development Manager/400 Concepts*, GC09-1377

  The *Application Development Manager/400 Concepts* provides a general discussion of the following:

  - What the Application Development Manager/400 product does, who its users are, and the advantages it provides
  - The relationships between this product and the AS/400 programs and the AD/Cycle* framework
  - The concepts relating to the tasks that application developers and project administrators perform.

- *SAA* AD/Cycle* Application Development Manager/400 Project Administrator's Guide*, SC09-1376

The *Application Development Manager/400 Project Administrator's Guide* provides information needed for Application Development Manager/400 project administrators to define project hierarchies for application development, and to define the creation and movement of parts in a project hierarchy by application developers.

Communications Manuals

These manuals provide background information for the communications APIs described in this book:

- *Communications and Systems Management Guide (Alerts and Distributed Systems Node Executive)*, SC41-9661

  The *Alerts and DSNX Guide* provides background information for the alert APIs.

- *Communications: Local Area Network Guide*, SC41-0004

  The *Local Area Network Guide* provides information about using an AS/400 system in a token-ring network, Ethernet network, or in a bridged environment.

- *Communications: Management Guide*, SC41-0024

  The *Communications Management Guide* provides information about working with communications status, errors, performance, line speed, and storage requirements.

- *Communications: Operating System/400* Communications Configuration Reference*, SC41-0001

  The *OS/400* Communications Configuration Reference* provides general communications configuration information about lines, controllers, devices, modes, class-of-service, configuration lists, network interfaces, and connection lists.

- *Communications: SNA Upline Facility Programmer's Guide*, SC41-9594

  The *SNA Upline Facility Programmer's Guide* provides information on display data streams using formatted buffers.

- *Communications: X.25 Network Guide*, SC41-0005

  The *X.25 Network Guide* provides information about using an AS/400 system in an X.25 packet-switched network.

- *Communications: 3270 Device Emulation Guide*, SC41-9602

  The *3270 Device Emulation Guide* provides a list of 3270 data stream commands, orders, and attributes that are supported.

The following IBM manuals are not specific to the AS/400 system, but do contain helpful communications information:

- *Advanced Function Printing: Data Stream Reference*, S544-3202, defines the AFP data stream used for advanced function page printing. It describes the use and syntax of data stream components and explains how the structured fields are interpreted in presenting composite pages of text, image, graphic, and bar code data. It specifies the approved content of the AFP data stream to be supported by IBM Print Services in each of the system environments. The AFP data stream is an architected presentation function set of the Mixed Object:Document Content Architecture, which is part of IBM's Systems Application Architecture.

- *IBM 8209 LAN Bridge Customer Information*, SA21-9994

  This *IBM 8209 LAN Bridge Customer Information* manual describes how to set up and use the 8209 Local Area Network Bridge.

- *Intelligent Printer Data Stream Reference*, S544-3417, describes the functions and services associated with IPDS. It is intended for systems programmers and other developers who need such information to develop or adapt a product or program to attach to an IPDS printer in an SAA communications network.

- *Print Services Facility User's Programming Guide for VM*, S544-3512, includes valid table reference codes and how these codes relate to the CHARS attribute and page definitions used in the spooled file APIs.

- *Print Services Facility/MVS Application Programming Guide*, S544-3084, includes valid table reference codes and how these codes relate to the CHARS attribute and page definitions used in the spooled file APIs.

- *Systems Network Architecture Formats*, GA27-3136

  The *SNA Formats* describes the contents of OS/400 alerts in detail.

- *Token-Ring Network Architecture Reference*, SC30-3374

- *Token-Ring Network Problem Determination Guide Kit*, SC30-3374

- *3270 Information Display System: 3274 Control Unit Description and Programmer's Guide*, GA23-0061

- *5250 Functions Reference Manual*, SA21-9247

The following manuals are not produced by IBM and are not specific to the AS/400 system, but do contain helpful communications information:

- *American National Standards Institute/Institute of Electrical and Electronics Engineers 802.2, 1985 - Logical Link Control, International Organization for Standardization/Draft International Standard 8802/2.*

- *American National Standards Institute/Institute of Electrical and Electronics Engineers 802.3, 1985 - Carrier Sense Multiple Access with Collision Detection, International Organization for Standardization/Draft International Standard 8802/3.*

- *American National Standards Institute/Institute of Electrical and Electronics Engineers 802.3a, b, d, c, 1988 -Supplements to Carrier Sense Multiple Access with Collision Detection American National Standards Institute/Institute of Electrical and Electronics Engineers Standard 802.3*, 1985.

- *American National Standards Institute/Institute of Electrical and Electronics Engineers 802.5, 1985 - Token Passing Ring, International Organization for Standardization/Draft International Standard 8802/5.*
- *The International Telegraph and Telephone Consultative Committee, Blue Book, Volume VIII - Fascicle VIII.2, Data Communications Networks: Services and Facilities, Recommendations X.1 - X.32, IXth Plenary Assembly*, Melbourne, November 14-25, 1988.

Manuals for DLS File System Users

Refer to these manuals for detailed information about the document library services (DLS) file system:

- *Document Interchange Architecture: Interchange Document Profile Reference*, SC23-0764

  The *IDP Reference* provides detailed information about the interchange document profile (IDP).

- *Office Services Concepts and Programmer's Guide*, SC41-9758

  The *Office Services Concepts and Programmer's Guide* provides background information for DLS file system users.

- *Systems Application Architecture* OfficeVision/400*: Managing OfficeVision/400*, SC41-9627

  The *Managing OfficeVision/400** describes administration procedures relevant to the DLS file system.

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Customer Satisfaction Feedback

Application System/400
System Programmer's
Interface Reference
Version 2

Publication No. SC41-8223-01

Overall, how would you rate this manual?

	Very Satisfied	Satisfied	Dissatisfied	Very Dissatisfied
Overall satisfaction				

How satisfied are you that the information in this manual is:

Accurate				
Complete				
Easy to find				
Easy to understand				
Well organized				
Applicable to your tasks				
THANK YOU!				

Please tell us how we can improve this manual:

_____

_____

_____

_____

_____

May we contact you to discuss your responses? __ Yes __ No

Phone: (____) _____ Fax: (____) _____

To return this form:

- Mail it
- Fax it
  United States and Canada: **800+937+3430**
  Other countries: **(+1)+507+253+5192**
- Hand it to your IBM representative.

Note that IBM may use or distribute the responses to this form without obligation.

Name _____ Address _____

Company or Organization _____ _____

Phone No. _____ _____

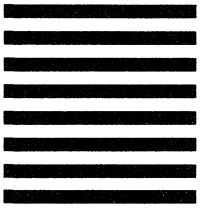
Customer Satisfaction Feedback
SC41-8223-01

IBM®

Cut c
Along

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DEPT 245
IBM CORPORATION
3605 HWY 52 N
ROCHESTER MN 55901-7899

Fold and Tape | **Please do not staple** | Fold and Tape

Cut o
Along

Program Number: 5738-SS1

Printed in Denmark by
Scanprint a-s, Viby J.

SC41-8223-01