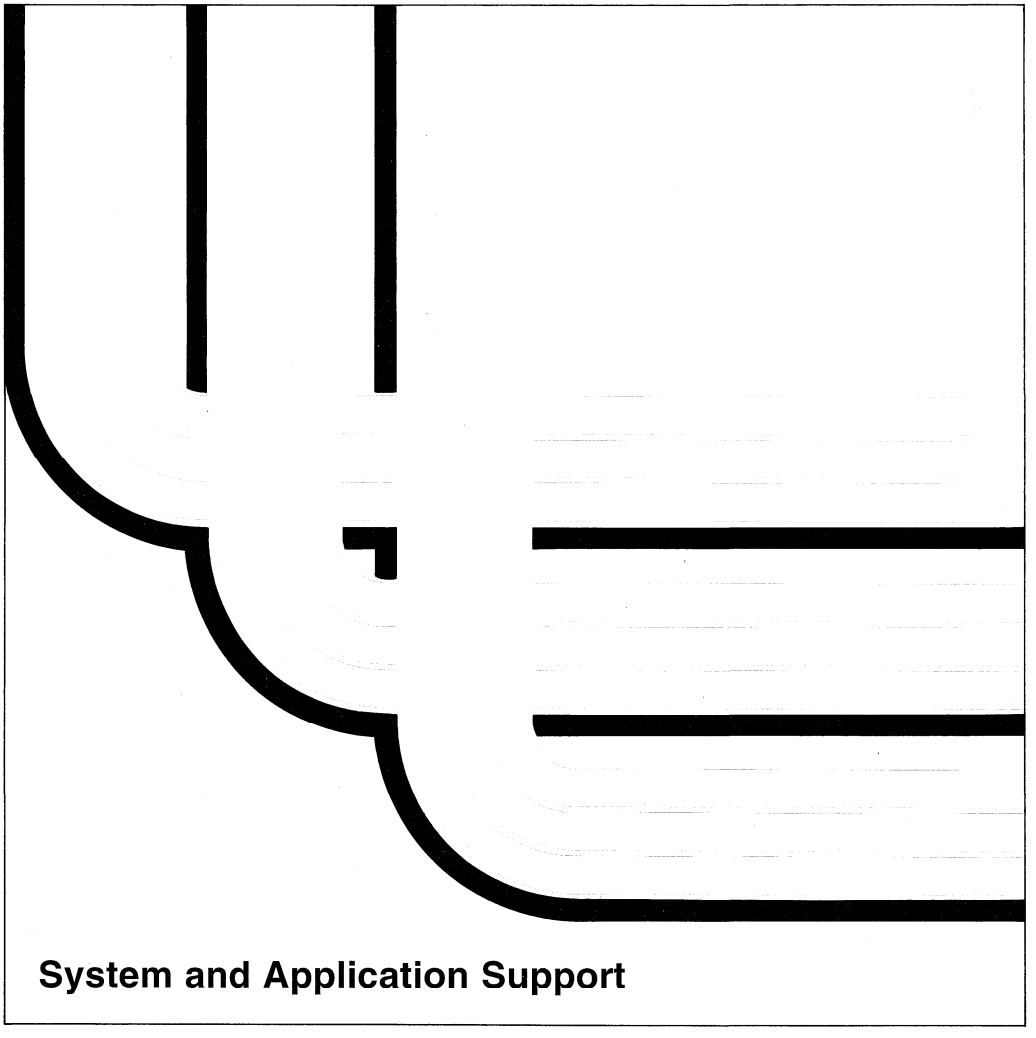
Application System/400

SC41-8223-02

System Programmer's Interface Reference Volume 1

Version 2



System and Application Support

Application System/400

SC41-8223-02

System Programmer's Interface Reference Volume 1

Version 2

Take Note!

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

Third Edition (November 1993)

This edition applies to the licensed program IBM Operating System/400, (Program 5738-SS1), Version 2 Release 3 Modification 0, and to all subsequent releases and modifications until otherwise indicated in new editions. This major revision makes obsolete SC41-8223-01. 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 1. Introduction to the OS/400 Callable APIs

Part 2. Communications APIs

Part 3. Configuration APIs

Part 4. Database File APIs

Part 5. Debugger APIs

Part 6. Dynamic Screen Manager APIs

Part 8. Hierarchical File System APIs

Part 9. High-Level Language APIs

Part 10. ILE CEE APIs

Part 11. Message Handling APIs

Part 12. National Language Support APIs

Part 13. Network Management APIs

Part 14. Object APIs

Part 15. Office APIs

Part 16. Operational Assistant APIs

Part 17. Performance Collector APIs

Part 18. Program and CL Command APIs

Part 19. Security APIs

Part 20. Software Product APIs

Part 21. Spooled File and Print APIs

Part 22. User Interface APIs

Part 23. Virtual Terminal APIs

Part 24. Work Management APIs

Part 25. Work Station Support APIs

Part 26. Miscellaneous APIs

Part 27. 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 that IBM 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 U.S.A.

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 xxxv 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 to make it easier for customer applications to interoperate with OS/400 system functions. If you have a requirement for additional interfaces to the OS/400 program, contact IBM at 1-507-253-1055 (FAX 507-253-2486).

Trademarks and Service Marks

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

- AD/Cycle
- Advanced Function Printing
- Advanced Peer-to-Peer Networking
- AFP
- Application System/400
- APPN
- AS/400
- BCOCA
- C/400
- CICS
- COBOL/400
- FORTRAN/400
- GDDM
- IBM
- ILE
- Integrated Language Environment
- Intelligent Printer Data Stream
- IPDS
- NetView
- OfficeVision
- OfficeVision/400
- Operating System/2
- Operating System/400
- Operational Assistant

OS/2
OS/400
Personal System/2
PS/2
PROFS
RPG/400
SAA
SQL/400
Systems Application Architecture
System/370
SystemView
VTAM
400

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

AT&T	AT&T
DCA	Digital Communications Associates
ETS	Siemens
Hewlett-Packard	Hewlett-Packard Company
LaserJet	Hewlett-Packard Company
PostScript	Adobe Systems Incorporated
RM/COBOL	Ryan McFarland Corporation
Telex	Telex Computer Products Inc
5ESS	AT&T

About This Manual

This manual describes the OS/400 application programming interfaces (APIs). Some APIs provide the same functions as control language (CL) commands and output file support. 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

The following changes have been made to this manual since Version 2 Release 2 of the OS/400 licensed program:

- The document library services information that was included in the hierarchical file systems (HFS) section has been moved to the *Office Services Concepts and Programmer's Guide*.
- The Character Data Representation Architecture APIs have been moved from this manual to the *Character Data Representation Architecture: Level 2 Reference*, SC09-1390.
- A list of APIs and exit programs by release has been added to Chapter 2. Refer to "APIs by Release" on page 1-3 for a list of all APIs in this manual.
- The User Object APIs part has been renamed to Object APIs.
- The following parts have been added to this manual:
  - Part 3, "Configuration APIs"
  - Part 5, "Debugger APIs"
  - Part 6, "Dynamic Screen Manager APIs"
  - Part 10, "ILE CEE APIs"
  - Part 12, "National Language Support APIs"
  - Part 17, "Performance Collector APIs"
  - Part 20, "Software Product APIs"
- Throughout this manual, APIs have been added and enhanced.

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 or output file support.
- 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 and will be optional 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.

The APIs will continue to work as they originally worked and any existing applications that use the APIs will continue to work without changes.

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:

Binding directory (*BNDDIR). An AS/400* object that contains a list of names of modules and service programs.

Module (*MODULE). An AS/400 object that is made up of the output of the compiler.

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

Service program (*SRVPGM). An object that packages externally supported callable routines into a separate object.

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 following sections:

- Outline the APIs described in this manual
- List the major operating system APIs described in other manuals
- List the APIs and exit programs by release.

APIs Described in This Manual

This manual presents the APIs in major functional categories, such as objects and work 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, "Configuration APIs." The APIs in this part get information about the configuration status of the system and about individual configuration descriptions.

Part 4, "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 5, "Debugger APIs." The APIs in this part can be used to write debuggers for the AS/400 system.

Part 6, "Dynamic Screen Manager APIs." The APIs in this part provide the screen I/O interfaces for Integrated Language Environment (ILE) high-level languages.

Part 7, "Edit Function APIs." The APIs in this part create and use edit masks.

Part 8, "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 9, "High-Level Language APIs." The APIs in this part communicate with compilers, and the SQL/400* and COBOL/400* languages.

Part 10, "ILE CEE APIs." The APIs in this part contain the CEE and CEE4 Integrated Language Environment (ILE) bindable APIs.

Part 11, "Message Handling APIs." The APIs in this part allow applications to work with AS/400 messages.

Part 12, "National Language Support APIs." The APIs in this part work with the national language support (NLS) function.

Part 13, "Network Management APIs." The APIs in this part handle alertable messages, work with problem logs, and use the SNA management services transport.

Part 14, "Object APIs." The APIs in this part create, manipulate, and delete user spaces, user indexes, and user queues. They also change, list, rename, and retrieve information about AS/400* objects.

Part 15, "Office 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 16, "Operational Assistant APIs." The APIs in this part provide access to the Operational Assistant* functions.

Part 17, "Performance Collector APIs." The APIs in this part describe the performance collector APIs and the performance monitor exit program.

Part 18, "Program and CL Command APIs." The APIs in this part create programs, retrieve program information, and retrieve command information.

Part 19, "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 20, "Software Product APIs." The APIs in this part work with software products and program temporary fixes (PTFs) on your system.

Part 21, "Spooled File and Print APIs." The APIs in this part retrieve information and manipulate spooled files.

Part 22, "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 23, "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 24, "Work Management APIs." The APIs in this part perform functions that are used in a wide variety of applications. These APIs retrieve and manipulate:

- Jobs
- Subsystem storage pools
- Subsystem job queues
- Data areas

- Network attributes
- System status
- System values
- Flight recorder

Part 25, "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 26, "Miscellaneous APIs." The APIs in this part retrieve information from main storage, remove bookmarks from a course, convert date and time, and list the contents of a save file.

Part 27, "Reference Information." This part includes an appendix containing programming examples, an appendix containing a chart of APIs and the public authority for each, 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:

  QCMDCHK Check Command Syntax
  QCMDEXC Execute Command
  QCLSCAN Scan for String Pattern
  QDCXLATE Translate Fields
  QCACHK Validity Check Command
  QPGMMENU Show Programmer Menu
  QCLRDTAQ Clear Data Queue
  QRCVDTAQ Receive Data Queue
  QMHQRDQD Retrieve Data Queue Description
  QMHRDQM Retrieve Data Queue Messages
  QSNDDTAQ Send Data Queue

- The REXX manuals describe the REXX APIs. See the *REXX/400 Programmer's Guide* for information about the following API:

  QREXQ REXX Queue Service

  See the *REXX/400 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.

APIs by Release

The following table shows the first release of the APIs and can be used if you are running multiple releases of OS/400 at your installation.

Figure 1-1 (Page 1 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Abnormal End (CEE4ABN)					X
Absolute Function (CEESxABS)					X
Add Commitment Resource (QTNADDCR)				X	
Add List Entry (QUIADDLE)				X	
Add List Multiple Entries (QUIADDLM)				X	
Add Pop-Up Window (QUIADDPW)				X	
Add Print Application (QUIADDPA)				X	
Add Product License Information (QLZADDLI)					X
Add User Index Entries (QUSADDUI)					X
Aid Spelling (QTWAIDSP)					X
Arccosine (CEESxACS)					X
Arcsine (CEESxASN)					X
Arctangent (CEESxATN)					X
Arctangent2 (CEESxAT2)					X
Backspace on Scroller Line (QsnSclBS)					X
Basic Random Number Generator (CEERANO)					X
Calculate Day of Week from Lilian Date (CEEDYWK)					X
Change COBOL Main Program (QLRCHGCM)				X	
Change Current Job (QWCCCJOB)					X
Change Directory Entry Attributes (QHFCHGAT)		X			

Figure 1-1 (Page 2 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Change Exception Message (QMHCHGEM)					X
Change File Pointer (QHFCHGFP)		X			
Change Library List (QLICHGLL)					X
Change Low-Level Environment (QsnChgEnv)					X
Change Mode Name (QNMCHGMN)				X	
Change Object Description (QLICOBJD)				X	
Change Office Program (QOGCHGOE)				X	
Change Pool Attributes (QUSCHGPA)	X				
Change Pool Tuning Information (QWCCHGTN)					X
Change Previous Sign-On Date (QSYCHGPR)					X
Change Session (QsnChgSsn)					X
Change User Password (QSYCHGPW)				X	
Change User Space (QUSCHGUS)	X				
Change User Space Attributes (QUSCUSAT)					X
Change Window (QsnChgWin)					X
Check Spelling (QTWCHKSP)					X
Check User Authority to an Object (QSYCUSRA)				X	
Check User Special Authorities (QSYCUSRS)				X	
Clear Buffer (QsnClrBuf)					X
Clear Field Table (QsnClrFldTbl)					X
Clear Screen (QsnClrScr)					X
Clear Scroller (QsnClrScl)					X
Clear Window (QsnClrWin)					X
Clear Window Message (QsnClrWinMsg)					X
Close Application (QUICLOA)				X	
Close Directory (QHFCLODR)		X			
Close Spooled File (QSPCLOSP)				X	
Close Stream File (QHFCLOSF)		X			
Close Virtual Terminal Path (QTVCLOVT)		X			
Conjugate of Complex (CEESxCJG)					X
Construct a Condition Token (CEENCOD)					X
Control File System (QHFCTLFS)		X			
Control Office Services (QOCCTLOF)				X	
Control Trace (QWTCTLTR)					X
Convert Authority Values to MI Value (QSYCVTA)				X	
Convert Date and Time Format (QWCCVTDT)		X			
Convert Date to Lilian Format (CEEDAYS)					X
Convert Edit Code (QECCVTEC)				X	
Convert Edit Word (QECCVTEW)				X	
Convert Integers to Seconds (CEEISEC)					X
Convert Lilian Date to Character Format (CEEDATE)					X
Convert Seconds to Character Timestamp (CEEDATM)					X
Convert Seconds to Integers (CEESECI)					X

Figure 1-1 (Page 3 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Convert Sort Sequence Table (QLGCNVSS)					X
Convert Timestamp to Number of Seconds (CEESECS)					X
Convert Type (QLICVTTP)				X	
Copy Buffer (QsnCpyBuf)					X
Copy Stream File (QHFCPYSF)		X			
Cosine (CEESxCOS)					X
Cotangent (CEESxCTN)					X
Create a Session (QsnCrtSsn)					X
Create a Window (QsnCrtWin)					X
Create Command Buffer (QsnCrtCmdBuf)					X
Create Directory (QHFCRTDR)		X			
Create Heap (CEECRHP)					X
Create Input Buffer (QsnCrtInpBuf)					X
Create Low-Level Environment (QsnCrtEnv)					X
Create Product Definition (QSZCRTPD)					X
Create Product Load (QSZCRTPL)					X
Create Program (QPRCRTPG)	X				
Create Program Temporary Fix (QPZCRTFX)					X
Create Spooled File (QSPCRTSP)		X			
Create User Index (QUSCRTUI)	X				
Create User Queue (QUSCRTUQ)	X				
Create User Space (QUSCRTUS)	X				
Decompose a Condition Token (CEEDCOD)					X
Define Heap Allocation Strategy (CEE4DAS)					X
Delete Buffer (QsnDltBuf)					X
Delete Directory (QHFDLTDR)		X			
Delete Field ID Definition (QsnDltFldId)					X
Delete List (QUIDLTL)				X	
Delete Low-Level Environment (QsnDltEnv)					X
Delete Product Definition (QSZDLTPD)					X
Delete Product Load (QSZDLTPL)					X
Delete Stream File (QHFDLTSF)		X			
Delete User Index (QUSDLTUI)		X			
Delete User Queue (QUSDLTUQ)		X			
Delete User Space (QUSDLTUS)		X			
Deregister Application (QNMDRGAP)				X	
Deregister APPN* Topology Information (QNMDRGTI)					X
Deregister File System (QHFDRGFS)		X			
Deregister Filter Notifications (QNMDRGFN)					X
Disable Link (QOLDLINK)		X			
Discard Heap (CEEDSHP)					X
Dispatch a Message (CEEMOUT)					X
Display Command Line Window (QUSCMDLN)	X				

Figure 1-1 (Page 4 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Display Directory Panels (QOKDSPDP)				X	
Display Help (QUHDSPH)		X			
Display Panel (QUIDSPP)				X	
Display Scroller Bottom (QsnDspSclB)					X
Display Scroller Top (QsnDspSclT)					X
Display Window (QsnDspWin)					X
Dump Flight Recorder (QWTDMPFR)				X	
Dump Lock Flight Recorder (QWTDMPLF)				X	
Edit (QECEDT)				X	
Enable Link (QOLELINK)		X			
End a Window (QsnEndWin)					X
End Application (QNMENDAP)				X	
End Data Stream Translation Session (QD0ENDTS)				X	
End Source Debug (QteEndSourceDebug)					X
Error Function and Its Complement (CEESxERF)					X
Exponential Base e (CEESxEXP)					X
Exponentiation (CEESxXPx)					X
Factorial (CEErSIFAC)					X
Filter Problem (QSXFTRPB)				X	
Find a Control Boundary (CEE4FCB)					X
Floating Complex Divide (CEESxDVD)					X
Floating Complex Multiply (CEESxMLT)					X
Force Buffered Data (QHFFRCSF)		X			
Free Storage (CEEFRST)					X
Gamma Function (CEESxGMA)					X
Generate a Beep (QsnBeep)					X
Generate Alert (QALGENA)		X			
Generate Program Temporary Fix Name (QPZGENNM)					X
Get a Message (CEEMGET)					X
Get AID (QsnGetAID)					X
Get Current Greenwich Mean Time (CEEGMT)					X
Get Current Local Time (CEELOCT)					X
Get Cursor Address (QsnGetCsrAdr)					X
Get Cursor Address with AID (QsnGetCsrAdrAID)					X
Get Dialog Variable (QUIGETV)				X	
Get Heap Storage (CEEGTST)					X
Get List Entry (QUIGETLE)				X	
Get List Multiple Entries (QUIGETLM)				X	
Get Offset from Universal Time Coordinated to Local Time (CEEGMTO)					X
Get Profile Handle (QSYGETPH)		X			
Get Space Status (QLYGETS)				X	
Get Spooled File Data (QSPGETSP)		X			
Get Stream File Size (QHFGETSZ)		X			

Figure 1-1 (Page 5 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Get String Information (CEEGSI)					X
Get Universal Time Coordinated (CEEUTC)					X
Get, Format, and Dispatch a Message (CEEMSG)					X
Go to Next Tab Position in Scroller Line (QsnSclTab)					X
Go to Start of Current Scroller Line (QsnSclCR)					X
Go to Start of Next Scroller Line (QsnSclNL)					X
Handle a Condition (CEE4HC)					X
Hyperbolic Arctangent (CEESxATH)					X
Hyperbolic Cosine (CEESxCSH)					X
Hyperbolic Sine (CEESxSNH)					X
Hyperbolic Tangent (CEESxTNH)					X
Imaginary Part of Complex (CEESxIMG)					X
Initialize Low-Level Environment Description (QsnInzEnvD)					X
Initialize Session Description (QsnInzSsnD)					X
Initialize Window Description (QsnInzWinD)					X
Insert Cursor (QsnInsCsr)					X
List Active Subsystems (QWCLASBS)		X			
List Authorized Users (QSYLAUTU)				X	
List Configuration Descriptions (QDCLCFGD)					X
List Database File Members (QUSLMBR)	X				
List Database Relations (QDBLDBR)				X	
List Fields (QUSLFLD)	X				
List ILE Program Information (QBNLPGMI)					X
List Job (QUSLJOB)	X				
List Job Log Messages (QMHLJOBL)					X
List Job Schedule Entries (QWCLSCDE)				X	
List Node List Entries (QFVLSTNL)					X
List Nonprogram Messages (QMHLSTM)					X
List Objects (QUSLOBJ)	X				
List Objects Secured by Authorization List (QSYLATLO)				X	
List Objects That Adopt Owner Authority (QSYLOBJP)				X	
List Objects User Is Authorized To or Owns (QSYLOBJA)				X	
List Performance Data (QPMLPFRD)					X
List Record Formats (QUSLRCD)	X				
List Registered File Systems (QHFLSTFS)		X			
List Save File (QSRLSAVF)					X
List Service Program Information (QBNLSPGM)					X
List Signed-On Users (QEZLSGNU)					X
List Spooled Files (QUSLSPL)	X				
List Subsystem Job Queues (QWDLSJBQ)		X			
List Users Authorized to Object (QSYLUSRA)				X	
Lock and Unlock Range in Stream File (QHFLULSF)		X			
Log Gamma Function (CEESxLGM)					X

Figure 1-1 (Page 6 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Log Program Temporary Fix Information (QPZLOGFX)					X
Log Software Error (QPDLOGER)					X
Logarithm Base e (CEESxLOG)					X
Logarithm Base 10 (CEESxLG1)					X
Logarithm Base 2 (CEESxLG2)					X
Map View Position (QteMapViewPosition)					X
Mark Heap (CEEMKHP)					X
Modular Arithmetic (CEESxMOD)					X
Move the Resume Cursor to a Return Point (CEEMRCR)					X
Move Program Messages (QMHMOVPM)			X		
Move Stream File (QHFMOVSF)		X			
Move Window (QsnMovWin)					X
Move Window by User (QsnMovWinUsr)					X
Nearest Integer (CEESxNIN)					X
Nearest Whole Number (CEESxNWN)					X
Open Directory (QHFOPNDR)		X			
Open Display Application (QUIOPNDA)				X	
Open Print Application (QUIOPNPA)				X	
Open Spooled File (QSPOPNSP)		X			
Open Stream File (QHFOPNSF)		X			
Open Virtual Terminal Path (QTVOPNVT)		X			
Operational Assistant Attention-Key-Handling (group jobs) (QEZMAIN)		X			
Operational Assistant Attention-Key-Handling (nongroup jobs) (QEZAST)		X			
Package Product Option (QSZPKGPO)					X
Pad between Two Screen Addresses (QsnWrtPadAdr)					X
Pad for N Positions (QsnWrtPad)					X
Positive Difference (CEESxDIM)					X
Print Panel (QUIPRTP)				X	
Print Scroller Data (QsnPrtScl)					X
Process Commands (QCAPCMD)					X
Process Extended Dynamic SQL (QSQPRCED)					X
Promote Message (QMHPRMM)					X
Put Command Buffer (QsnPutBuf)					X
Put Command Buffer and Perform Get (QsnPutGetBuf)					X
Put Dialog Variable (QUIPUTV)				X	
Put Input Command (QsnPutInpCmd)					X
Put Output Command (QsnPutOutCmd)					X
Put Spooled File Data (QSPPUTSP)		X			
Put Window Message (QsnPutWinMsg)					X
Query (QQQQRY)				X	
Query Century (CEEQCEN)					X
Query Color Support (QsnQryColorSup)					X
Query Display Mode Support (QsnQryModSup)					X

Figure 1-1 (Page 7 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Query If Scroller in Line Wrap Mode (QsnQrySclWrp)					X
Query Keyboard Buffering (QWSQRYWS)		X			
Query Line Description (QOLQLIND)		X			
Query 5250 (QsnQry5250)					X
Read from Stream File (QHFRDSF)		X			
Read from Virtual Terminal (QTVRDVT)		X			
Read Build Information (QLYRDBI)				X	
Read Data from Session (QsnReadSsnDta)					X
Read Directory Entries (QHFRDDR)		X			
Read Immediate (QsnReadImm)					X
Read Input Fields (QsnReadInp)					X
Read Modified Alternate (QsnReadMDTAlt)					X
Read Modified Fields (QsnReadMDT)					X
Read Modified Immediate Alternate (QsnReadMDTImmAlt)					X
Read Screen (QsnReadScr)					X
Reallocate Storage (CEECZST)					X
Receive Data (QNMRCVDT)				X	
Receive Data (QOLRECV)		X			
Receive Nonprogram Message (QMHRCVM)			X		
Receive Operation Completion (QNMRCVOC)				X	
Receive Program Message (QMHRCVPM)			X		
Register a User-Written Condition Handler (CEEHDLR)					X
Register Activation Group Exit Procedure (CEE4RAGE)					X
Register Application (QNMREGAP)				X	
Register APPN Topology Information (QNMRGTI)					X
Register Call Stack Entry Termination User Exit Procedure (CEERTX)					X
Register Debug View (QteRegisterDebugView)					X
Register File System (QHFRGFS)		X			
Register Filter Notifications (QNMRGFN)					X
Release Heap (CEERLHP)					X
Release License (QLZARLS)					X
Release Profile Handle (QSYRLSPH)		X			
Remove All Bookmarks from a Course (QEARMVBM)				X	
Remove Commitment Resource (QTNRMVCR)				X	
Remove Debug View (QteRemoveDebugView)					X
Remove List Entry (QUIRMVLE)				X	
Remove Nonprogram Messages (QMHRMVM)			X		
Remove Pop-Up Window (QUIRMVPW)				X	
Remove Print Application (QUIRMVPA)				X	
Remove Program Messages (QMHRMVPM)			X		
Remove User Index Entries (QUSRMVUI)					X
Rename Directory (QHFRNMDR)		X			
Rename Object (QLIRNMO)					X

Figure 1-1 (Page 8 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Rename Stream File (QHFRNMSF)		X			
Request License (QLZAREQ)					X
Resend Escape Message (QMHRSNEM)			X		
Resize Window (QsnRszWin)					X
Resize Window by User (QsnRszWinUsr)					X
Restore Screen (QsnRstScr)					X
Retrieve Alert (QALRTVA)				X	
Retrieve AID Code on Read (QsnRtvReadAID)					X
Retrieve Buffer Data Length (QsnRtvBufLen)					X
Retrieve Buffer Size (QsnRtvBufSiz)					X
Retrieve Command Information (QCDRCMDI)				X	
Retrieve Commitment Information (QTNRCMTI)				X	
Retrieve Configuration Status (QDCRCFGS)					X
Retrieve Controller Description (QDCRCTLD)					X
Retrieve Current Window (QsnRtvCurWin)					X
Retrieve Cursor Address on Read (QsnRtvReadAdr)					X
Retrieve COBOL Error Handler (QLRRTVCE)				X	
Retrieve Data Area (QWCRDTAA)					X
Retrieve Debug Attribute (QteRetrieveDeubgAttribute)					X
Retrieve Device Description (QDCRDEVD)					X
Retrieve Directory Entry Attributes (QHFRTVAT)		X			
Retrieve Display Mode (QsnRtvMod)					X
Retrieve Field Information (QsnRtvFldInf)					X
Retrieve File Description (QDBRTVFD)				X	
Retrieve ILE Version and Platform ID (CEEGPID)					X
Retrieve Job Description Information (QWDRJOBD)				X	
Retrieve Job Information (QUSRJOBI)	X				
Retrieve Job Queue Information (QSPRJOBQ)				X	
Retrieve Language ID (QLGRTVLI)					X
Retrieve Length of Data in Input Buffer (QsnRtvDtaLen)					X
Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen)					X
Retrieve License Information (QLZARTV)					X
Retrieve Line Description (QDCRLIND)					X
Retrieve List Attributes (QUIRTVLA)				X	
Retrieve Low-Level Environment Description (QsnRtvEnvD)					X
Retrieve Low-Level Environment User Data (QsnRtvEnvDta)					X
Retrieve Low-Level Environment Window Mode (QsnRtvEnvWinMod)					X
Retrieve Main Storage (QVTRMSTG)				X	
Retrieve Member Description (QUSRMBRD)	X				
Retrieve Message (QMHRTVM)			X		
Retrieve Mode Name (QNMRTVMN)				X	
Retrieve Module Views (QteRetrieveModuleViews)					X
Retrieve Network Attributes (QWCRNETA)					X

Figure 1-1 (Page 9 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Retrieve Nonprogram Message Queue Attributes (QMHRMQAT)					X
Retrieve Number of Bytes Read from Screen (QsnRtvReadLen)					X
Retrieve Number of Columns to Shift Scroller (QsnRtvSclNumShf)					X
Retrieve Number of Fields Read (QsnRtvFldCnt)					X
Retrieve Number of Rows to Roll Scroller (QsnRtvSclNumRoll)					X
Retrieve Object Description (QUSROBJD)	X				
Retrieve Office Programs (QOGRTVOE)				X	
Retrieve Operational Descriptor Information (CEEDOD)					X
Retrieve Output Queue Information (QSPROUTQ)				X	
Retrieve Pointer to Field Data (QsnRtvFldDta)					X
Retrieve Pointer to User Space (QUSPTRUS)	X				
Retrieve Product Information (QSZRTVPR)					X
Retrieve Program Associated Space (QCLRPGAS)					X
Retrieve Program Information (QCLRPGMI)				X	
Retrieve Program Temporary Fix Information (QPZRTVFX)					X
Retrieve Program Variable (QTERTVPV)					X
Retrieve Read Information (QsnRtvReadInf)					X
Retrieve Registered Filters (QNMRRGF)					X
Retrieve Request Message (QMHRTVRQ)				X	
Retrieve Screen Dimensions (QsnRtvScrDim)					X
Retrieve Service Program Information (QBNRSPGM)					X
Retrieve Session Data (QsnRtvSsnDta)					X
Retrieve Session Description (QsnRtvSsnD)					X
Retrieve Session Input Line to Command Line (QsnRtvSsnLin)					X
Retrieve Sort Sequence Table (QLGRTVSS)					X
Retrieve Spooled File Attributes (QUSRSPLA)	X				
Retrieve Stopped Position (QteRetrieveStoppedPosition)					X
Retrieve Subsystem Information (QWDRSBSD)		X			
Retrieve System Status (QWCRSSTS)					X
Retrieve System Values (QWCRSVAL)					X
Retrieve User Authority to Object (QSYRUSRA)				X	
Retrieve User Index Attributes (QUSRUIAT)					X
Retrieve User Index Entries (QUSRTVUI)					X
Retrieve User Information (QSYRUSRI)				X	
Retrieve User Space (QUSRTVUS)	X				
Retrieve User Space Attributes (QUSRUSAT)					X
Retrieve View Text (QteRetrieveViewText)					X
Retrieve Window Data (QsnRtvWinDta)					X
Retrieve Window Description (QsnRtvWinD)					X
Retrieve Writer Information (QSPRWTRI)					X
Return the Relative Invocation Number (CEE4RIN)					X
Return Default Date String for Country (CEEFMDA)					X
Return Default Date/Time String for Country (CEEFMDT)					X

Figure 1-1 (Page 10 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Return Default Time String for Country (CEEFMTM)					X
Roll Down (QsnRollDown)					X
Roll Scroller Down (QsnRollSclDown)					X
Roll Scroller Up (QsnRollSclUp)					X
Roll Up (QsnRollUp)					X
Save Information (QEZSAVIN)		X			
Save Screen (QsnSavScr)					X
Scan String for Mixed Data (QLGSCNMX)					X
Send Alert (QALSNDA)		X			
Send Break Message (QMHSNDBM)			X		
Send Data (QOLSEND)		X			
Send Error (QNMSNDER)				X	
Send Message (QEZSNDMG)		X			
Send Nonprogram Message (QMHSNDM)			X		
Send Program Message (QMHSNDPM)			X		
Send Reply (QNMSNDRP)				X	
Send Reply Message (QMHSNDRM)			X		
Send Request (QNMSNDRQ)				X	
Send Request for OS/400 Function (QTVSNDRQ)		X			
Set Century (CEESCEN)					X
Set Current Window (QteSetCurWin)					X
Set Cursor Address (QsnSetCsrAdr)					X
Set COBOL Error Handler (QLRSETCE)				X	
Set Error State (QsnSetErr)					X
Set Field (QsnSetFld)					X
Set Filter (QOLSETF)		X			
Set Keyboard Buffering (QWSSETWS)		X			
Set List Attributes (QUISETLA)				X	
Set Lock Flight Recorder (QWTSETLF)				X	
Set Low-Level Environment Window Mode (QsnSetEnvWinMod)					X
Set Output Address (QsnSetOutAdr)					X
Set Profile (QWTSETP)		X			
Set Screen Image (QUISETSC)				X	
Set Space Status (QLYSETS)				X	
Set Stream File Size (QHFSETSZ)		X			
Set Timer (QOLTIMER)		X			
Set Trace (QWTSETTR)					X
Set Window Services Attributes (QsnSetWinAtr)					X
Shift Scroller Left (QsnShfSclL)					X
Shift Scroller Right (QsnShfSclR)					X
Signal a Condition (CEESGL)					X
Signal the Termination-Imminent Condition (CEETREC)					X
Sine (CEESxSIN)					X

Figure 1-1 (Page 11 of 11). APIs by Release

API	V1R3	V2R1	V2R1.1	V2R2	V2R3
Sort (QLGSORT)					X
Sort Input/Output (QLGSRTIO)					X
Square Root (CEESxSQT)					X
Start a Window (QsnStrtWin)					X
Start Application (QNMSTRAP)				X	
Start Data Stream Translation Session (QD0STRTS)				X	
Start New Scroller Line at Current Position (QsnSclLF)					X
Start New Scroller Page (QsnSclFF)					X
Start Source Debug (QteStartSourceDebug)					X
Store Program Associated Space (QCLSPGAS)					X
Submit Debug Command (QteSubmitDebugCommand)					X
Tangent (CEESxTAN)					X
Test for Omitted Argument (CEETSTA)					X
Toggle between Line Wrap/Truncate Mode (QsnTglSclWrp)					X
Transfer of Sign (CEESxSGN)					X
Transform AFP* to ASCII (QWPZTAFP)					X
Translate Data Stream (QD0TRNDS)				X	
Truncate Character Data (QLGTRDTA)					X
Truncation (CEESxINT)					X
Unregister a User Conditional Handler (CEEHDLU)					X
Unregister Call Stack Entry Invocation Termination User Exit Procedure (CEEUTX)					X
Update List Entry (QUIUPDLE)				X	
Validate Language ID (QLGVLID)					X
Work with Collector (QPMWKCOL)					X
Work with Jobs (QEZBCHJB)		X			
Work with Messages (QEZMSG)		X			
Work with Printer Output (QEZOUTPT)		X			
Work with Problem (QPDWRKPB)				X	
Write to Display (QsnWTD)					X
Write to Stream File (QHFWRTSF)		X			
Write to Virtual Terminal (QTVWRTVT)		X			
Write Build Information (QLYWRTBI)				X	
Write Characters to Scroller (QsnWrtSclChr)					X
Write Data (QsnWrtDta)					X
Write Line to Scroller (QsnWrtSclLin)					X
Write Structured Field Major (QsnWrtSFMaj)					X
Write Structured Field Minor (QsnWrtSFMin)					X
Write Transparent Data (QsnWrtTDta)					X

Exit Programs by Release

The following table shows the first release of the exit programs and can be used if you are running multiple releases of OS/400 at your installation.

Figure 1-2. Exit Programs by Release

Exit Program	V2R1	V2R2	V2R3
Action List Option and Pull-Down Field Choice		X	
Application Formatted Data		X	
Change Directory Entry Attributes (QHFCHGAT)	X		
Change File Pointer (QHFCHGFP)	X		
Close Directory (QHFCLODR)	X		
Close Stream File (QHFCLOSF)	X		
Commit and Rollback		X	
Control File System (QHFCTLFS)	X		
Copy Stream File (QHFCPYSF)	X		
Create Directory (QHFCRTDR)	X		
Cursor-Sensitive Prompt		X	
Customized Separator Page			X
Debug Session Handler			X
Delete Directory (QHFDLTDR)	X		
Delete Stream File (QHFDLTSF)	X		
Directory Search		X	
Directory Supplier			X
Directory Verification		X	
Document Conversion		X	
Document Handling		X	
End Job Session	X		
Error Handling		X	
Force Buffered Data (QHFFRCSF)	X		
Function Key		X	
General Panel Checking		X	
Get Stream File Size (QHFGETSZ)	X		
Incomplete List		X	
Lock and Unlock Range in Stream File (QHFLULSF)	X		
Menu Item		X	
Move Stream File (QHFMOVSF)	X		
Open Directory (QHFOPNDR)	X		
Open Stream File (QHFOPNSF)	X		
Performance Monitor			X
Program Stop Handler			X
Program Temporary Fix		X	
QLPUSER			X
Read Directory Entries (QHFRDDR)	X		
Read from Stream File (QHFRDSF)	X		
Rename Directory (QHFRNMDR)	X		
Rename Stream File (QHFRNMSF)	X		
Retrieve Directory Entry Attributes (QHFRTVAT)	X		
Set Stream File Size (QHFSETSZ)	X		
Software Program Functions			X
Start Job Session	X		
Tailoring Automatic Cleanup	X		
Tailoring Operational Assistant Backup		X	
Tailoring Power Off	X		
Write to Stream File (QHFWRTSF)	X		

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 considerations
- 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
- System-domain versus user-domain object considerations

Language Selection Considerations

Other than the Integrated Language Environment* (ILE) APIs, which only use the ILE C/400* language, 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
ILE C/400*	X	X	X	X	X[2]	X[2]	X	X	X	X
CL		X[7]	X[7]	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	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	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.
7. Unless otherwise stated, the %BIN function exists on the Change Variable (CHGVAR) CL command.

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
- Omitted 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	OPxxxxxxxx[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 the 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, or unpredictable results may occur.

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 *NAME (basic 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 also referred to as pass-by-reference.) 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

1 The *NAME format is a character string that must begin with an alphabetic character (A through Z, $, #, or @ followed by up to 9 characters (A through Z, 0 through 9, $, #, @, _, or .).

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.

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.

Omitted Parameters

The Integrated Language Environment (ILE) APIs may have parameters that can be omitted. When these parameters are omitted, you must pass a null pointer.

Manipulating a User Space with Pointers

Some languages, such as C/400, COBOL/400, System C/400 PRPQ, Pascal, and PL/I, support pointers. Pointers allow you to manipulate information more rapidly from the user space.[2] 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 location 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.

[2] A **user space** is an object consisting of a collection of bytes that can be used for storing any user-defined information.

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-       3/22/90  11:02:56         SIZE-       0000400
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-

 0000E0 5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C  5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C ************************************
      LINES   000200     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 40F14040 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-       3/22/90  11:02:56          SIZE-       0000400
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                                *
 000040  5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C  5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C **********************************
           LINES 000060   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

                                        ***** END OF LISTING *****
```

Area changed after using the Change User Space (QUSCHGUS)

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..64.) 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:

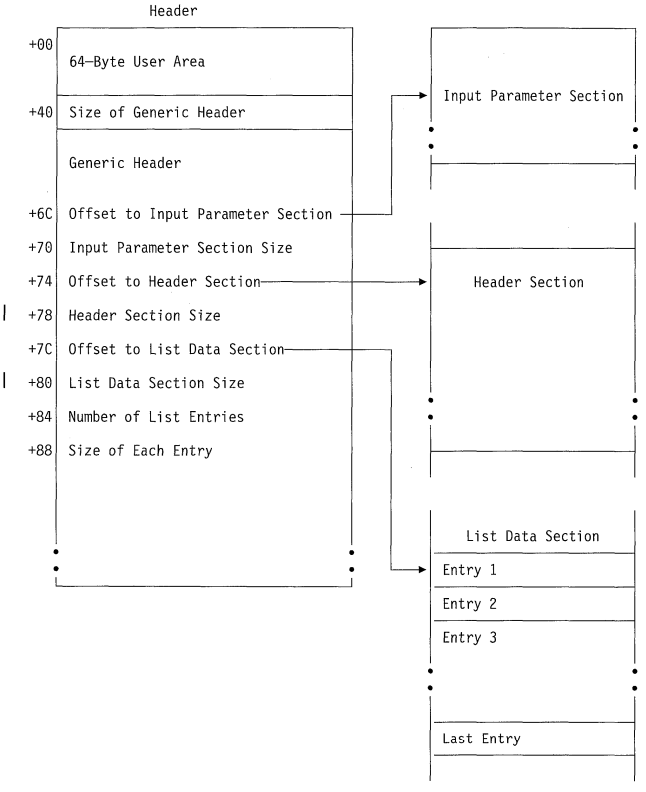
```
H* ****************************************************************
H* ****************************************************************
H*                                                                 *
H*  PROGRAM:  CHANGUS                                              *
H*                                                                 *
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 USERAREA
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:



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 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 list data section.

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

C Complete and accurate.
I Incomplete. The information you received is not accurate or complete.
P Partial but accurate. The information you received is accurate, but the API had more information to return than the user space could hold. See "Partial List Considerations" on page 2-11 for more information about partial lists.

Number of list 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				
Dec	**Hex**	**Use**	**Type**	**Field**
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 error information available to the API to return, in bytes. If this is 0, 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
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                                          17 272 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 63-24.

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 be able to return more information to the application than fits in the user space. The information returned is correct, but not complete. If the information is not complete, both of the following occur:

- A P is returned in the information status field of the generic user space layout; refer to "General Data Structure" on page 2-7.
- The API supports a continuation handle.

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. If the API does not support a continuation handle, you need to call the API again and use more restrictive values for the parameters.

System-Domain versus User-Domain Object Considerations

You should consider the following when working with user objects. (**User objects** are user spaces, user queues, and user indexes.) Prior to Version 2 Release 3 Modification 0, all user objects were created into the user domain. Starting in Version 2 Release 3 Modification 0, user objects can exist in either the user domain or the system domain. The allow user domain (QALWUSRDMN) system value determines which libraries can contain user-domain user objects. The default QALWUSRDMN system value is set to *ALL, but can be changed by system administrators on individual machines to be one library or a list of libraries. If your application requires direct access to user-domain user objects in a library that is not specified in the QALWUSRDMN value, your system administrator can add the library to the system value.

System-domain user objects operate differently from user-domain user objects at security levels 40 and 50. At those levels, you cannot directly manipulate system-domain user objects. You must use APIs to work with the objects.

Note: On a system configured for C2[3] system security, QALWUSRDMN is set to QTEMP (only the QTEMP library can contain user-domain user objects).

For more information about C2 security, refer to the *Guide to Enabling C2 Security* manual.

[3] **C2** is a level of security defined in the *Trusted Computer System Evaluation Criteria* (TCSEC) published by the United States Government.

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, "Using the User-Defined Communications Programs for File Transfer" on page A-34 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.

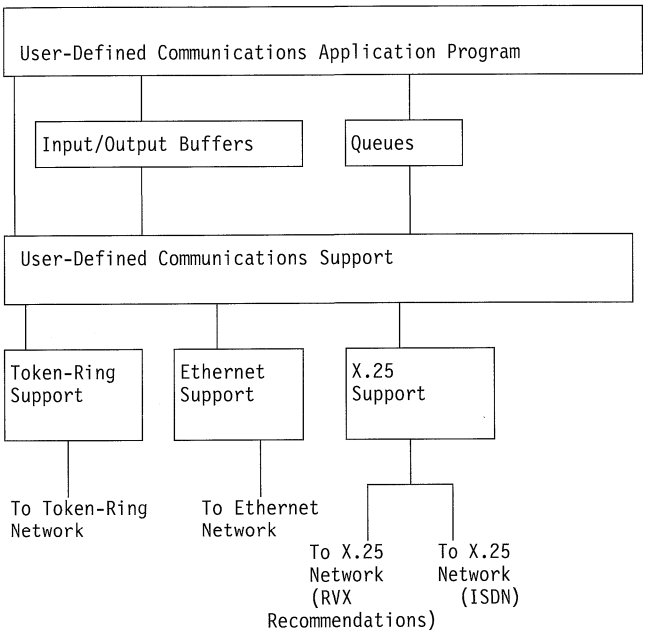
Overview

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 43, "User Space APIs" on page 43-1 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.

Terminology

Listed below are terms that are important in understanding the information contained in this part.

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.

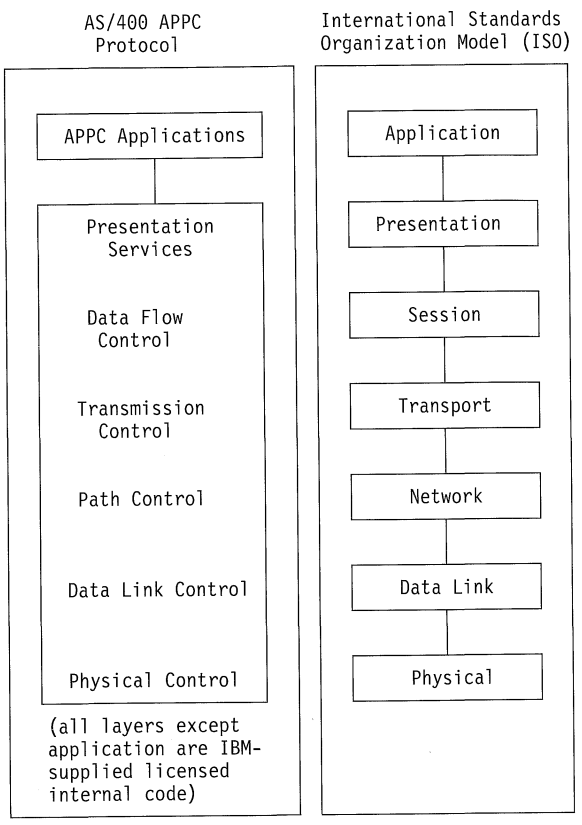


Figure 3-2. AS/400 APPC versus ISO Model

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).

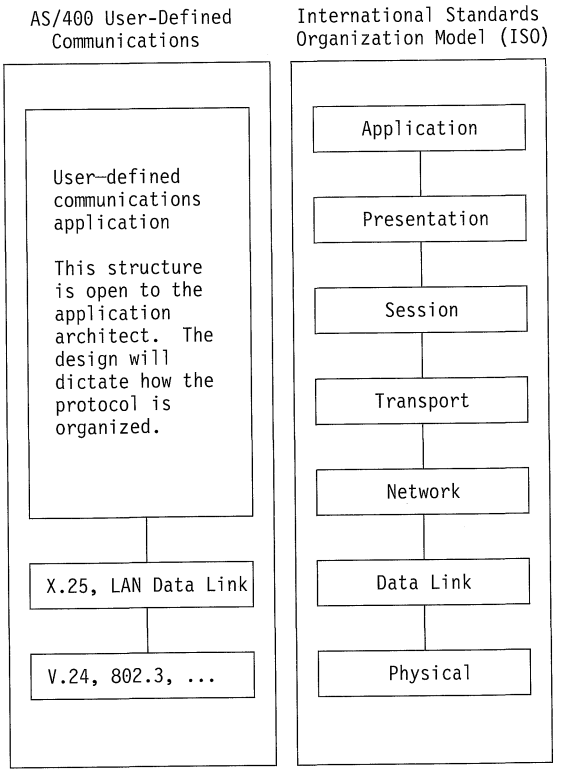


Figure 3-3. AS/400 User-Defined versus ISO Model

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.

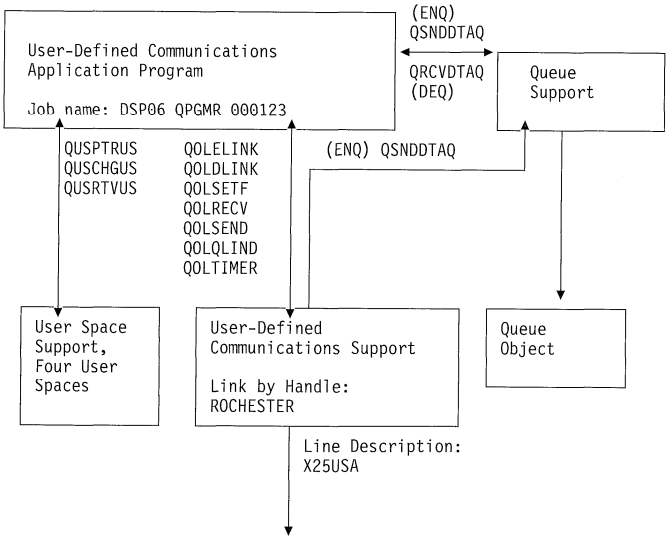


Figure 4-1. Overview of API Relationships

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.

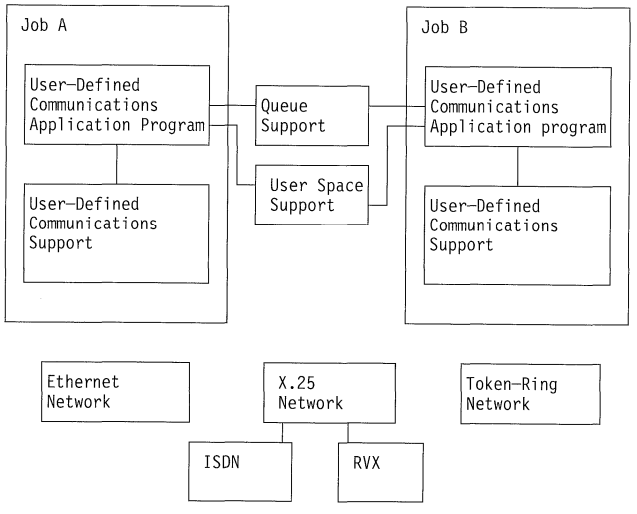


Figure 4-2. Application Programming Interface to Job Structure

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 4-17) 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

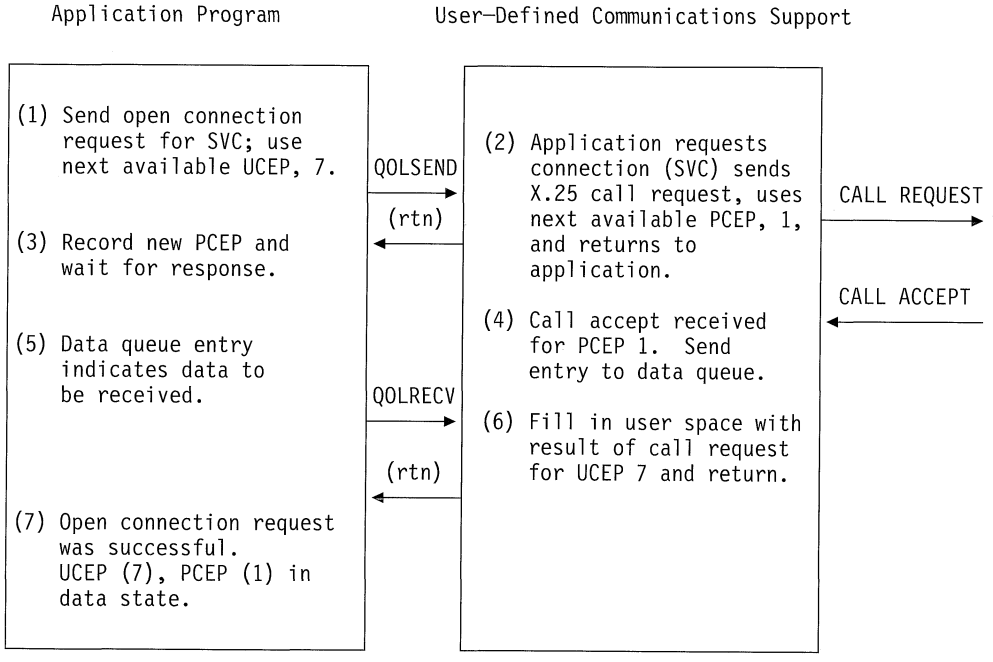


Figure 4-3. Example 1: Normal Connection Establishment

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

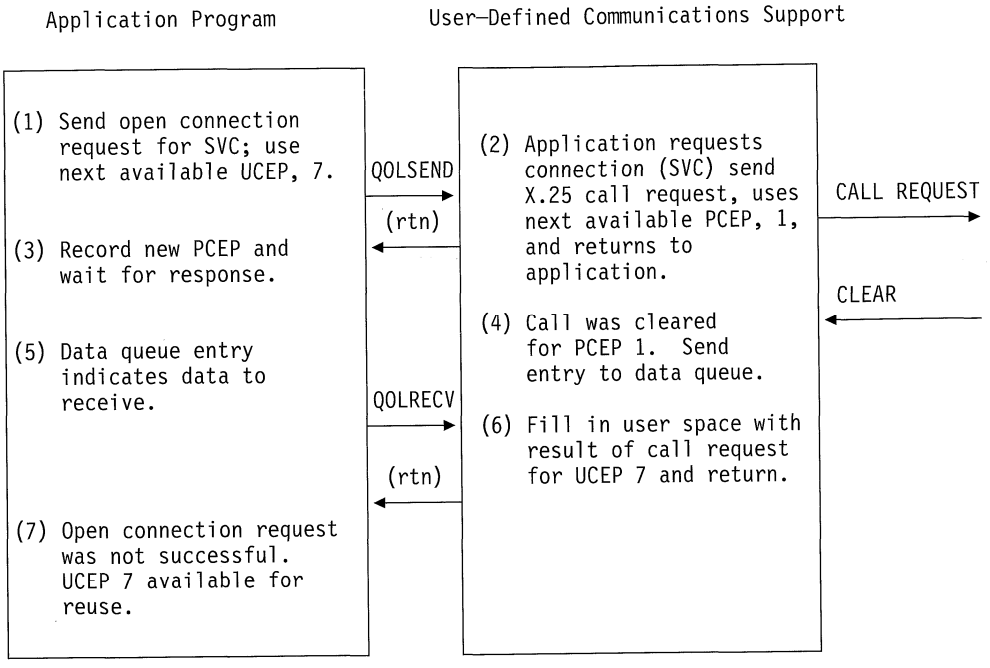


Figure 4-4. Example 2: Connection Request Cleared by Network/Remote System

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

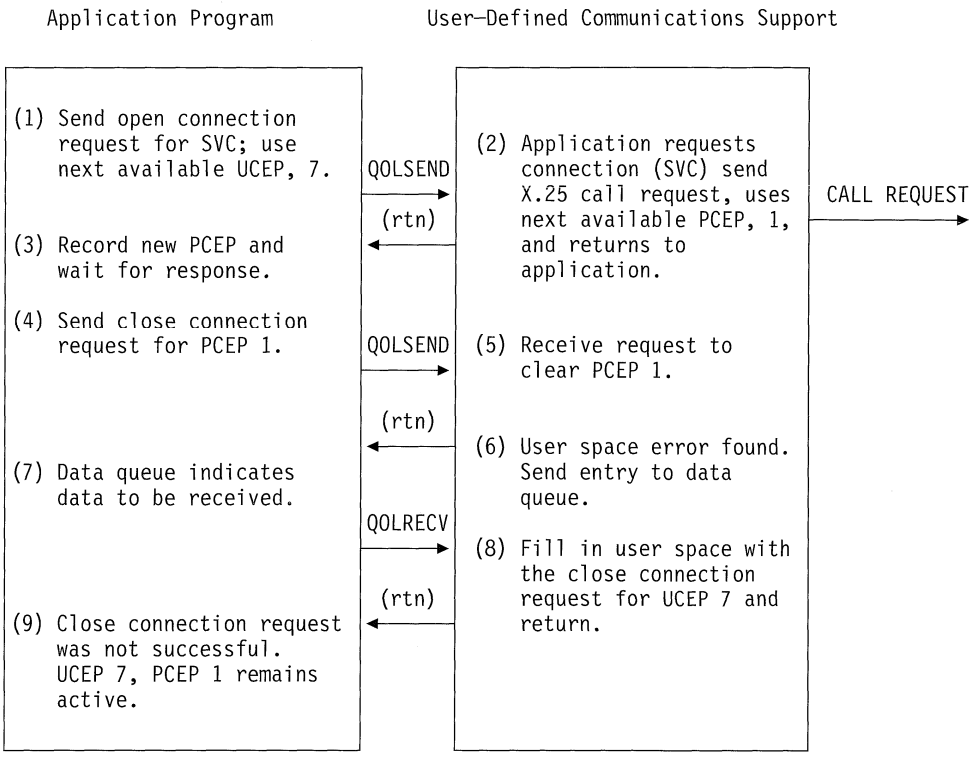


Figure 4-5. Example 3: Request to Clear Connection with Outstanding Call (Unsuccessful)

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

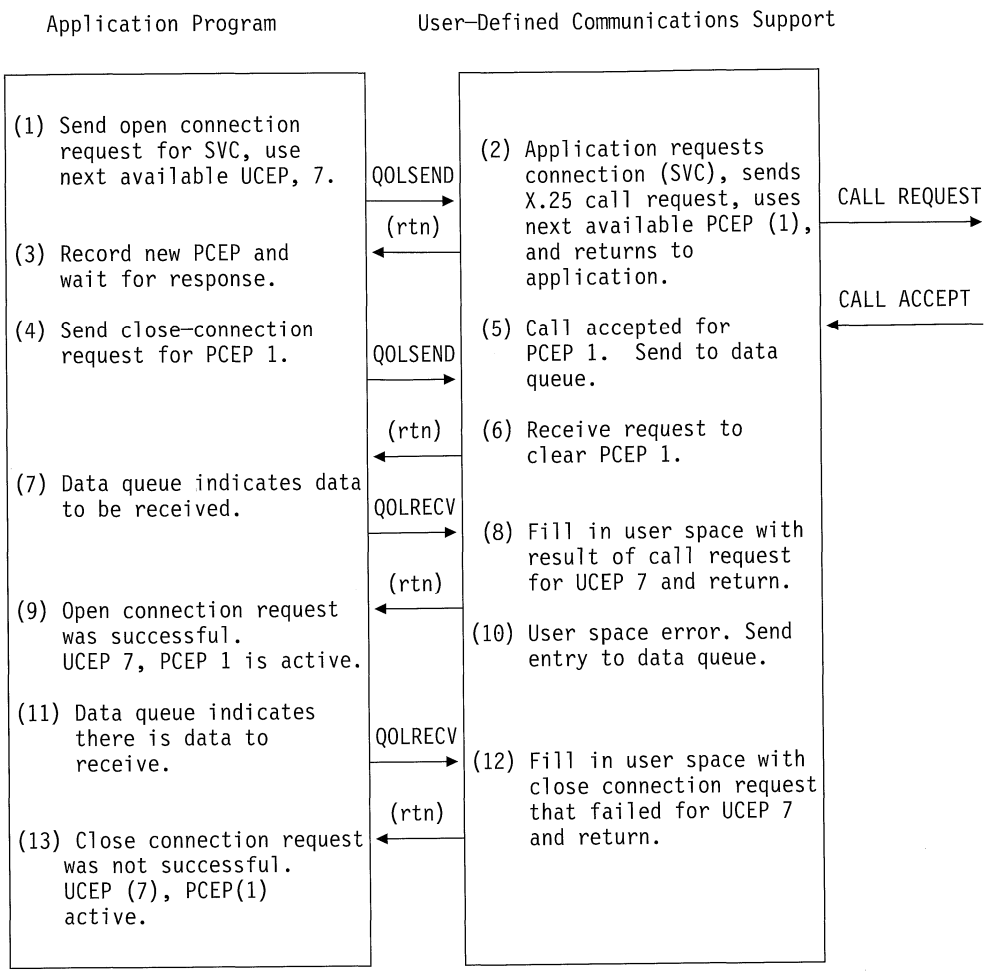


Figure 4-6. Unsuccessful Attempt to Clear Outstanding (Successful) Call

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

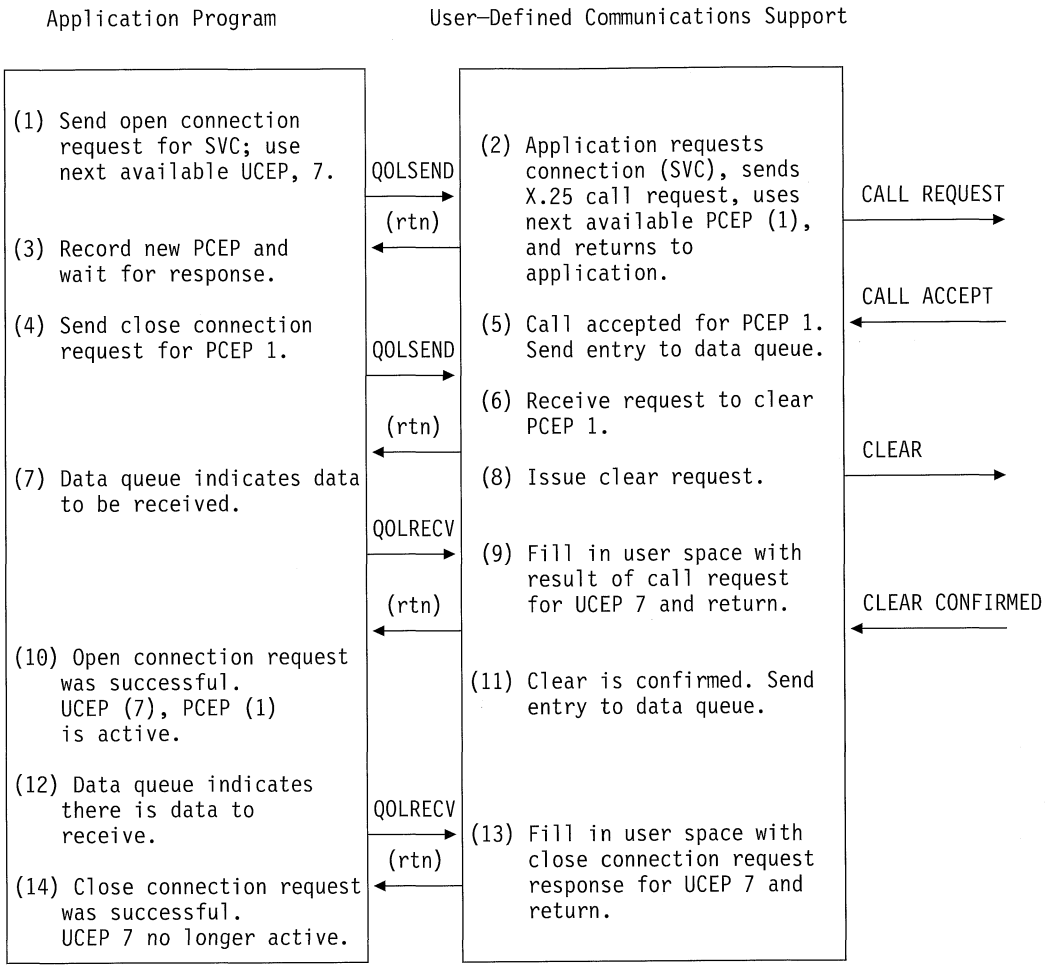


Figure 4-7. Example 5: Successful Attempt to Clear Outstanding (Successful) Call

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

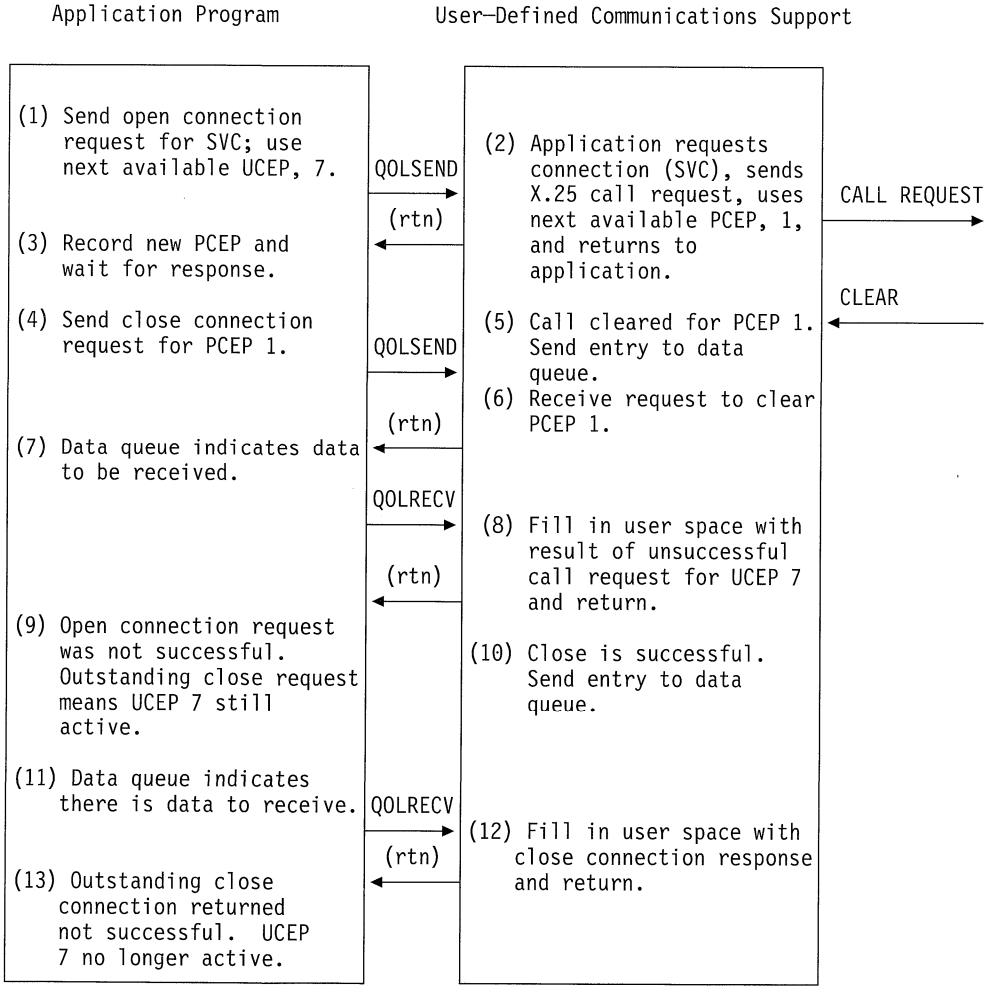


Figure 4-8. Example 6: Successful Attempt to Clear Outstanding (Unsuccessful) Call

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

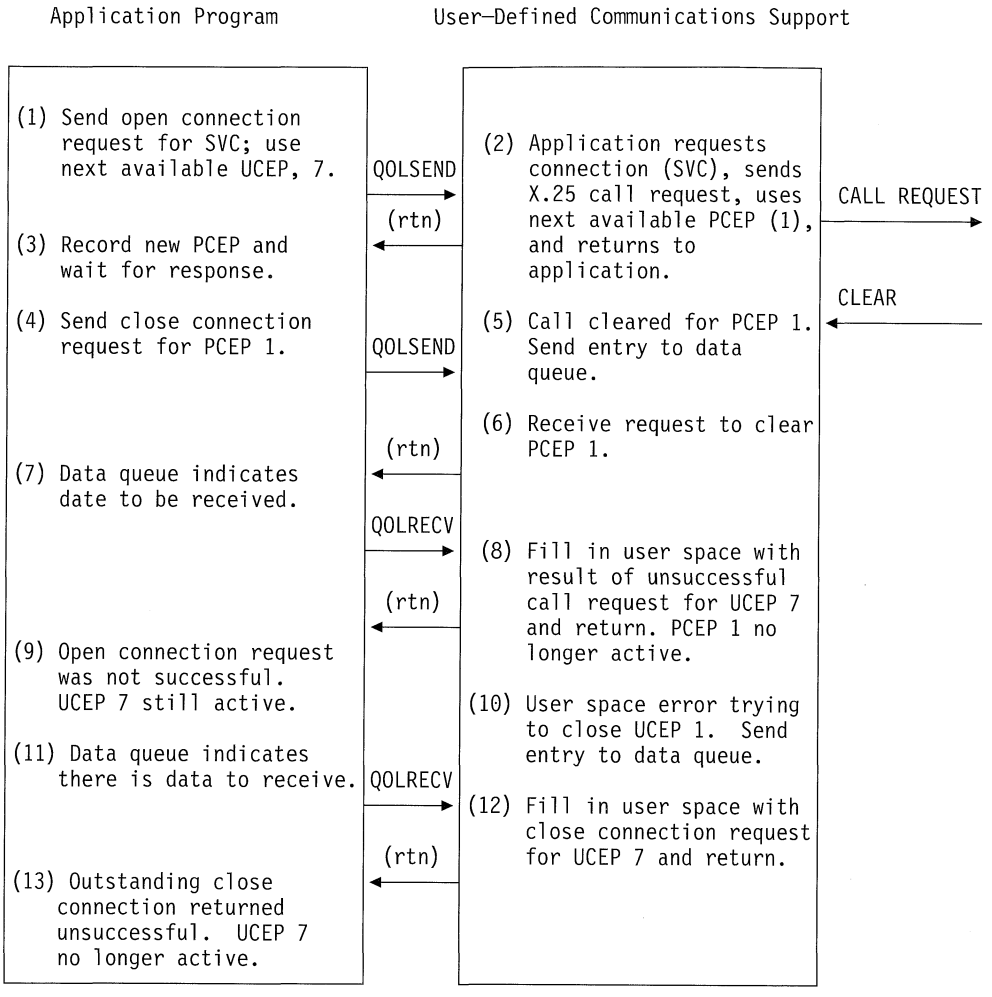


Figure 4-9. Example 7: Unsuccessful Attempt to Clear Outstanding (Unsuccessful) Call

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.

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

Example 1

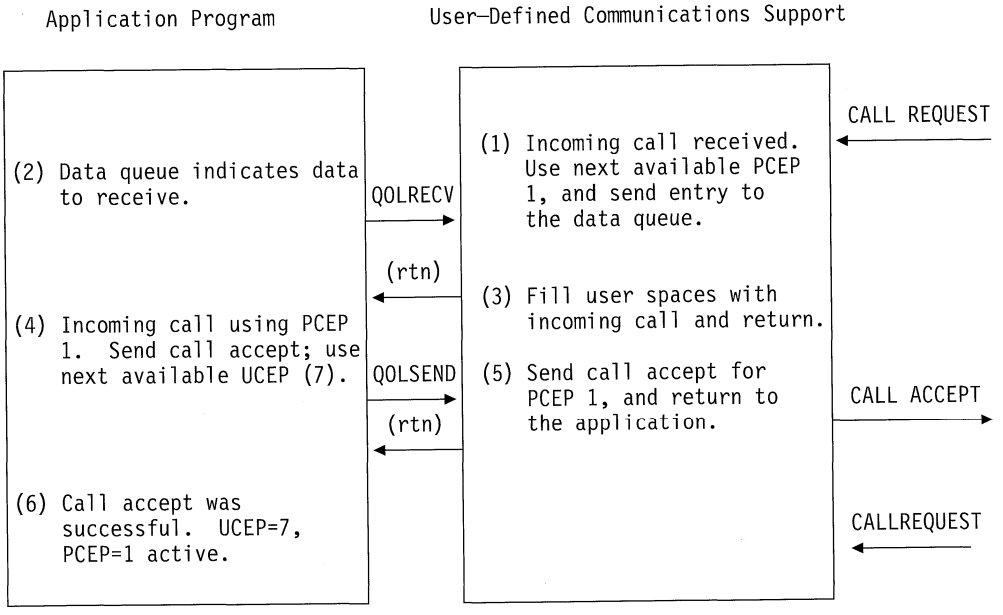


Figure 4-10. Example 1: Normal Connection Establishment

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 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

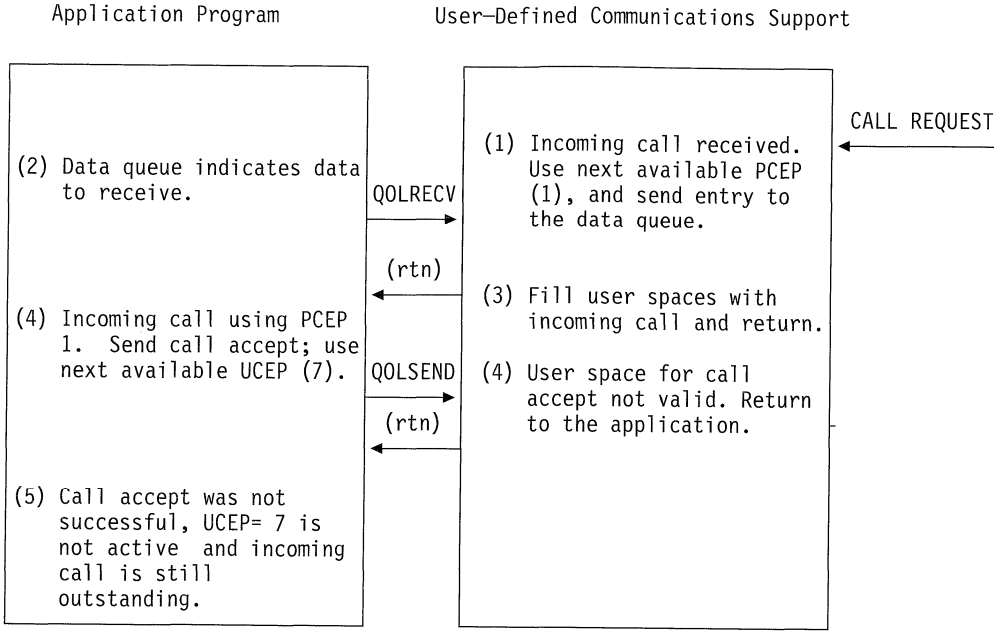


Figure 4-11. Example 2: Send Call Accept Not Valid

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

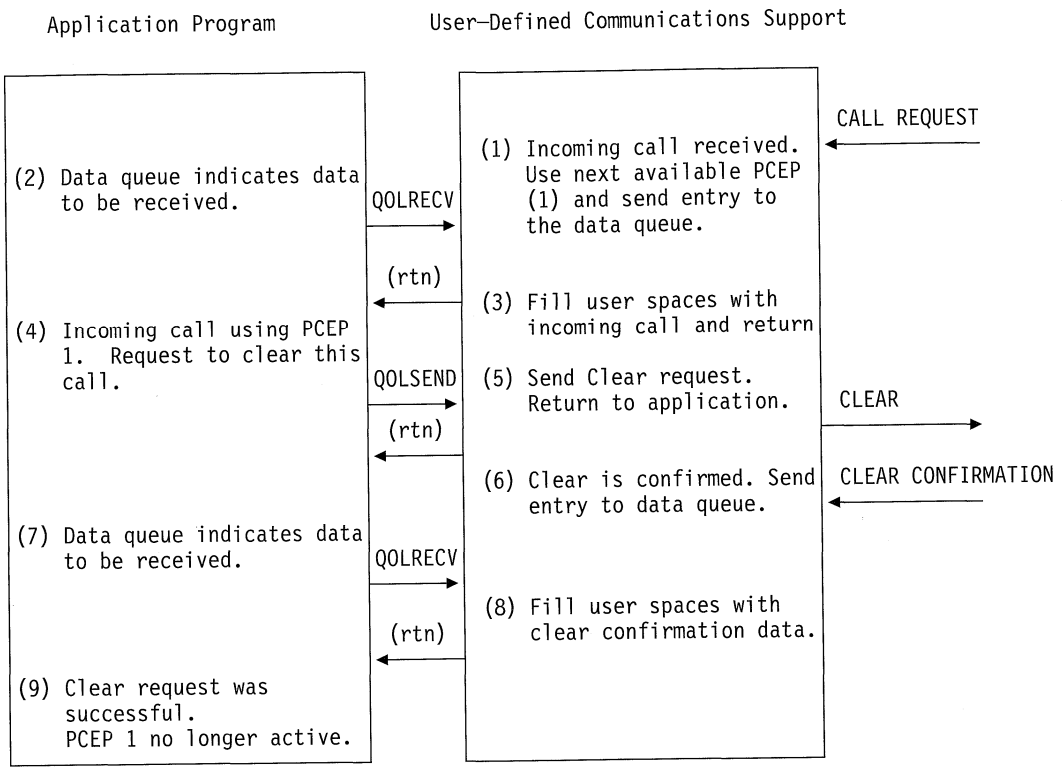


Figure 4-12. Example 3: Send Clear for Incoming Call

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

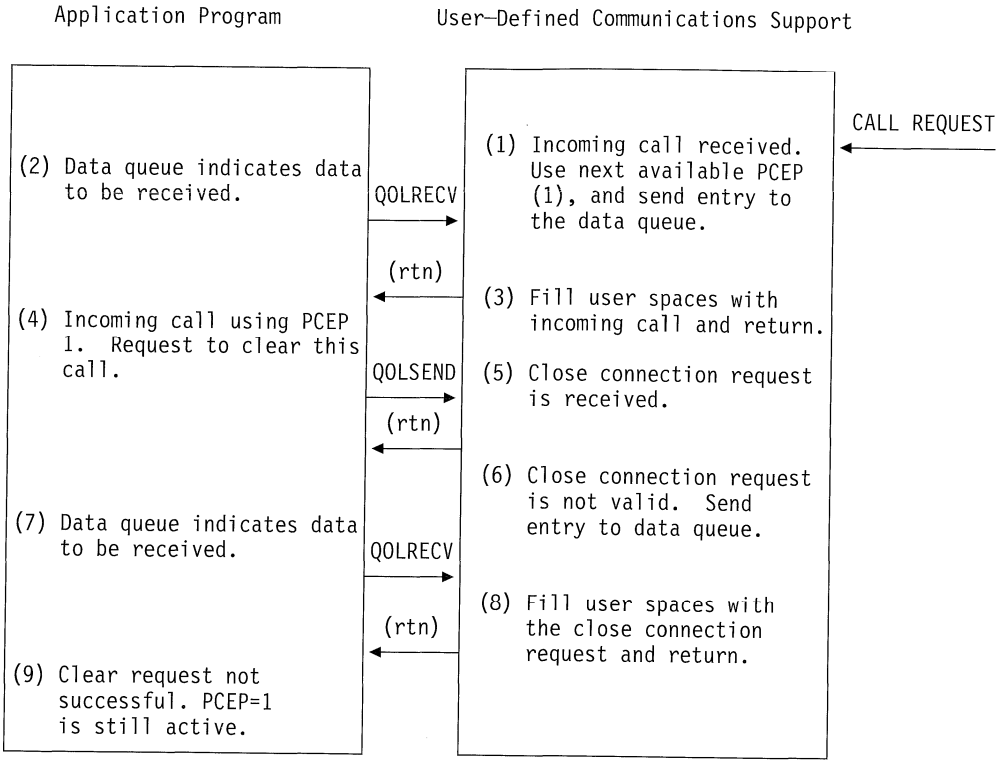


Figure 4-13. Example 4: Send Clear for Incoming Call

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

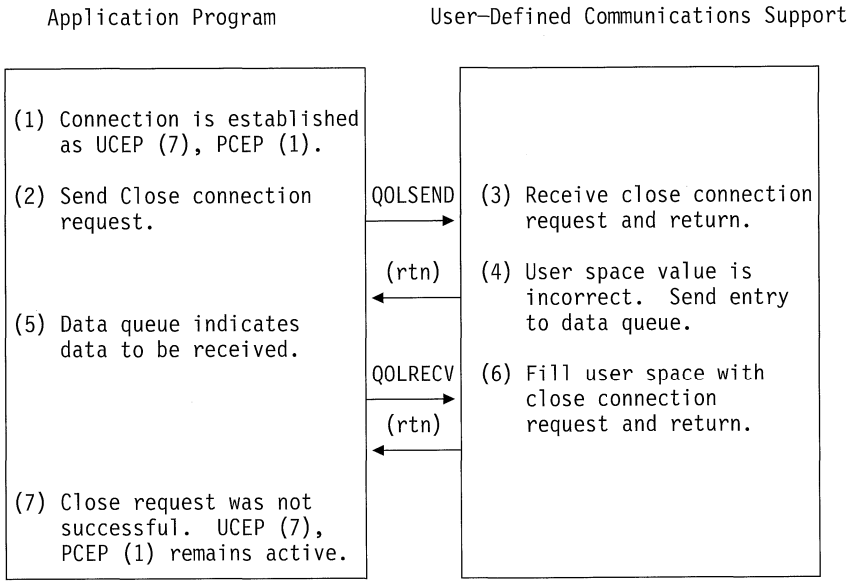


Figure 4-14. Example 1: Close Connection Request Is Not Valid

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

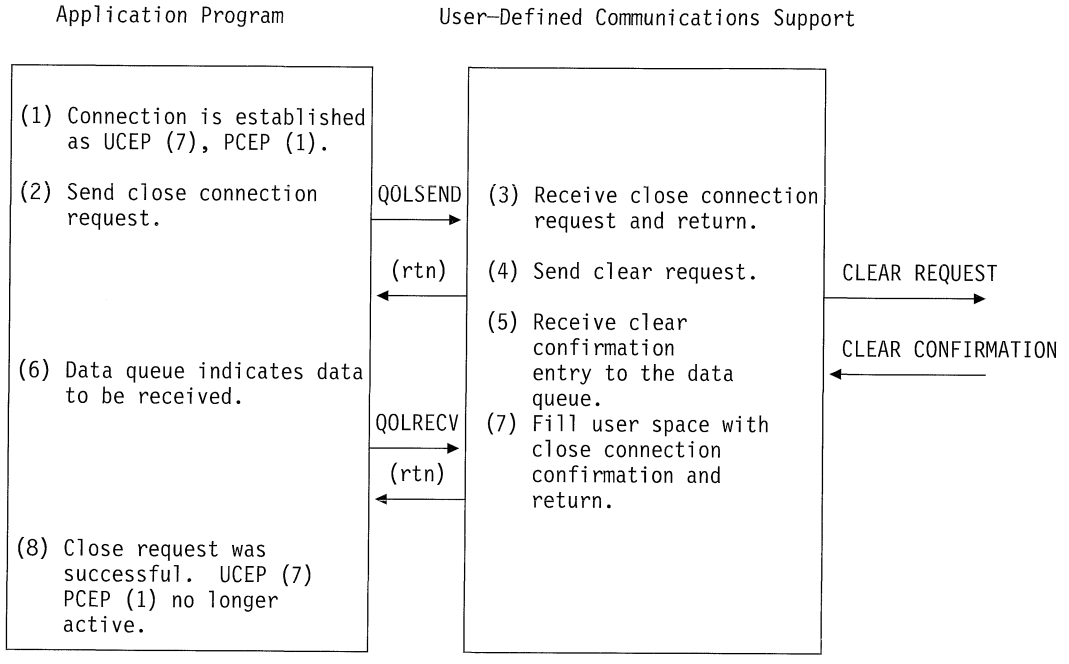


Figure 4-15. Example 2: Close Connection Request Is Valid

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	
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

Packet Type	Application Input or Access	SVC	PVC	N/A
Reject (REJ)	**Note:** This packet is not necessarily available on all networks and is not supported by the AS/400 system.			X
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 6-30 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" 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 on page 4-26, 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.

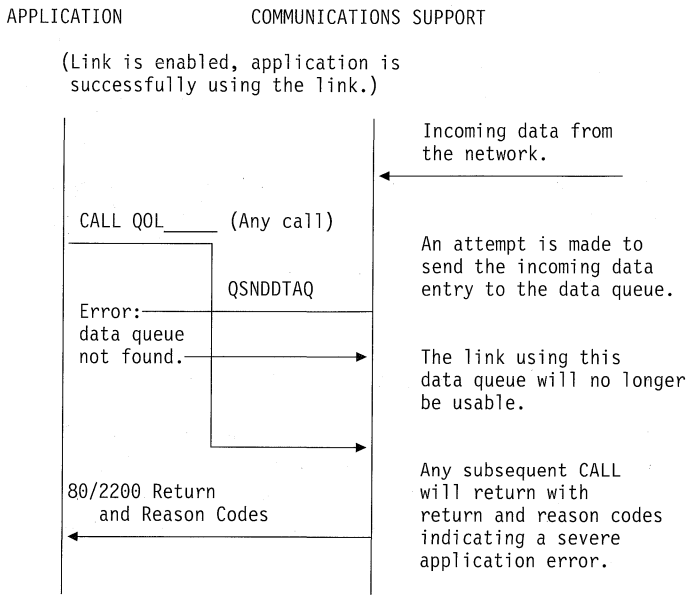


Figure 4-19. Using the Data Queue

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.

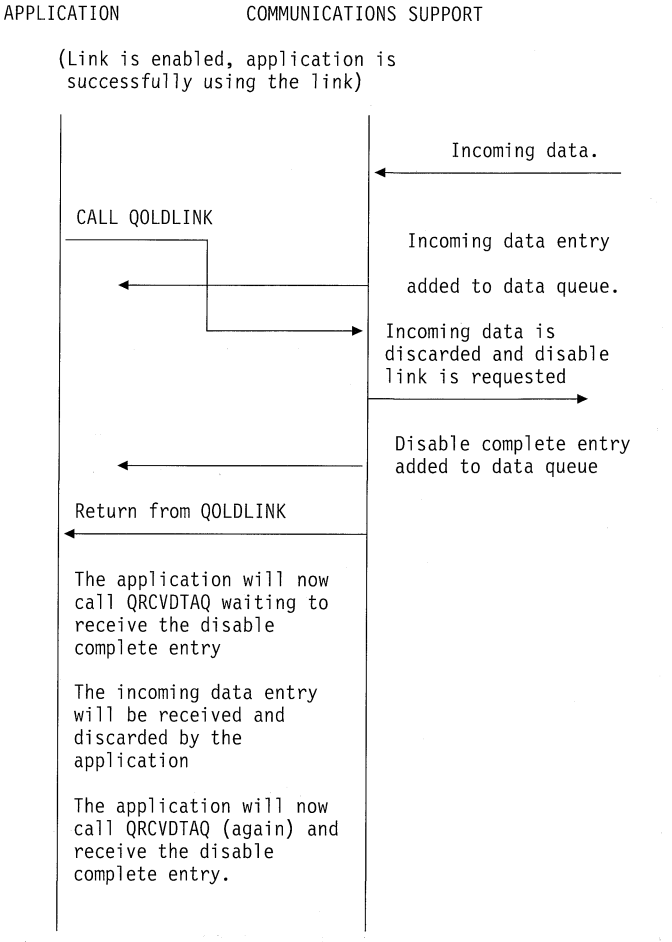


Figure 4-20. Application Disables the Link

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.

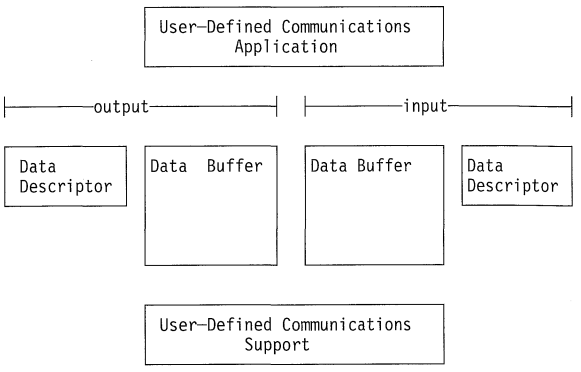


Figure 4-21. User Spaces

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.

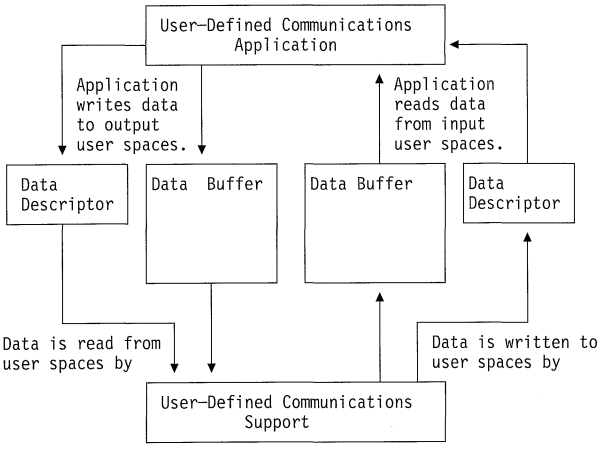


Figure 4-22. Input/Output Operations

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 6-26 and "Receive Data (QOLRECV) API" on page 6-13 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.
  - CPF9872 Program &1 in library &2 ended. Reason code &3.
- 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.

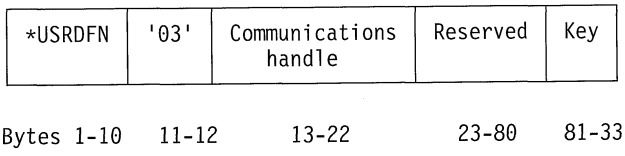


Figure 5-5. Incoming-Data Entry

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.

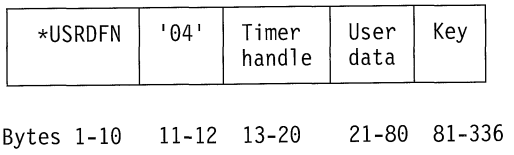


Figure 5-6. Timer-Expired Entry

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."

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

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.
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.

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

Return / Reason Code	Meaning	Recovery
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Query Line Description (QOLQLIND) API

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)

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.

Required Parameter Group

Return code
OUTPUT; BINARY(4)
The recovery action to take. See "Return and Reason Codes" on page 6-12.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-12.

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 on page 6-6.

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. 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.
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.

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

Field	Type	Description
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.
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
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.		

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

Field	Type	Description
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.
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.

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

Field	Type	Description
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 (Page 1 of 2). 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.

Figure 6-9 (Page 2 of 2). Return and Reason Codes for the QOLQLIND API

Return / Reason Code	Meaning	Recovery
83/2007	Line description damaged.	Delete and re-create the line description. Try the request again.

Error Messages

CPF91F0 E Internal system error.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 6-15 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 6-17 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 6-22.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-22.

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 6-22 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.

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

Field	Type	Description
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'.
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. 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 6-42 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 6-42 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').
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 6-42 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.

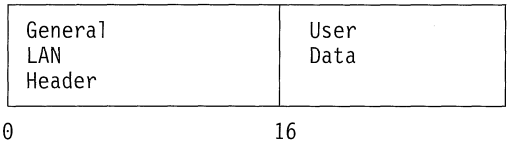


Figure 6-12. Ethernet 802.3 and Token-Ring Frames with No Routing Information

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 6-42 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 on page 6-17 shows the fields and offsets used for token-ring frames with routing information.

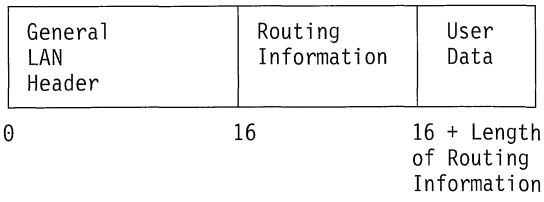


Figure 6-13. Token-Ring Frames with Routing Information

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.

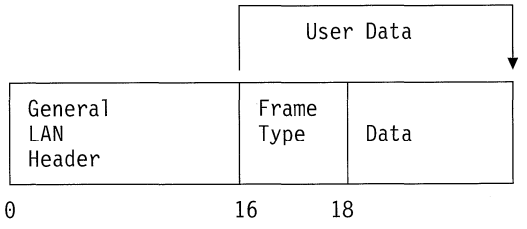


Figure 6-14. Ethernet Version 2 Frames

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).

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

Operation	Meaning
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. 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 6-26 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.
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 6-26 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]

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

Field	Type	Description
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.
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 6-26 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.

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

Field	Type	Description
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.
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 (Page 1 of 2). 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.

Figure 6-22 (Page 2 of 2). Format of Data for X'B301' Operation

Field	Type	Description
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 (Page 1 of 2). 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.

Figure 6-23 (Page 2 of 2). Return and Reason Codes Indicating No Data Received

Return / Reason Code	Meaning	Recovery
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 (Page 1 of 2). 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.

Figure 6-29 (Page 2 of 2). Return and Reason Codes for X.25 Operation X'B111'

Return / Reason Code	Meaning	Recovery
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.

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

Return / Reason Code	Meaning	Recovery
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).
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 6-28 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 6-30 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 6-38.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-38.

Diagnostic data
OUTPUT; CHAR(40)
Additional diagnostic data. See "Diagnostic Data Parameter Format" on page 6-27 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 6-13 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 6-13 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 6-29 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 6-13 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 6-13 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 6-13 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 6-13 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 6-31 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 6-13 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'.
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.

Figure 6-39 (Page 2 of 3). Format of Data for X'B000' Operation (Initiate an SVC Call)

Field	Type	Description
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.
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 6-13 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 more than once.

Figure 6-39 (Page 3 of 3). Format of Data for X'B000' Operation (Initiate an SVC Call)

Field	Type	Description
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 6-13 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. Format of Data for X'B100' Operation

Field	Type	Description
Reserved	CHAR(2)	This field must be set to hexadecimal zeros.
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 6-13 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 6-13 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 6-13 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 6-13 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 6-31 and "X.25 Operation X'B400'" on page 6-36 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.

Figure 6-45 (Page 2 of 2). Return and Reason Codes for X.25 Operation X'0000'

Return / Reason Code	Meaning	Recovery
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.	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. 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.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 6-43 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

[1] All active filters for a link must be of the same type.

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 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 6-45.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-45.

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 (Page 1 of 2). 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.

Figure 6-51 (Page 2 of 2). Filter Header Data

Field	Type	Description
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. 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.
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. 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.
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 6-26 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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 6-48.

Reason code
OUTPUT; BINARY(4)
The error that occurred. See "Return and Reason Codes" on page 6-48.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 4-24 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 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 7-8. 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 4-28.

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)

Figure 7-11 (Page 2 of 2). Error Codes Reported on X'B001', X'B301', and X'B400' Operations

Error Code	Description	Cause
3200 338C	Response restricted by fast select	User data is not allowed with restriction
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
- Translate from a 5250 read screen format to a 3270 read buffer format
- Translate from a 5250 read screen with extended attributes to a 3270 read buffer format

The following figures show the translation options available when your application calls the data stream translation APIs.

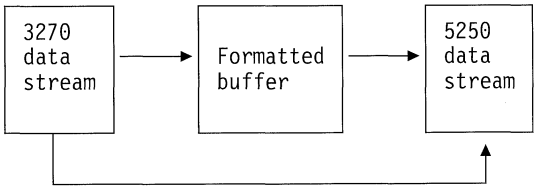


Figure 8-1. Translations for Output Operations

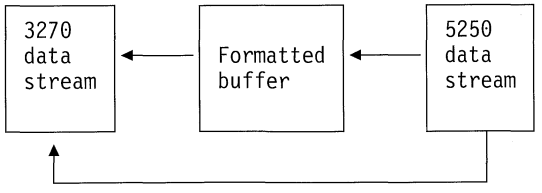


Figure 8-2. Translations for Input Operations

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.
- The display does not support extended attributes.

Note: The full capabilities of the device can be determined only if a 5250 query has been sent to the device.

The 5252 query is sent the first time a user signs on after the device is varied on. The results remain in effect until the device is varied off. If no one has signed on since the device was varied on, some of the characteristics will default to those assumed for *BASIC display devices.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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
3270RB	Create a 3270 data stream for the data stream that is expected in response to a 3270 Read Buffer command
*FORMAT	Create a formatted buffer for the data. See the *SNA Upline Facility Programmer's Guide* to determine the format of the buffer header.

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
5250RS	Contains a 5250 data stream that results from a 5250 Read Screen command
5250RSE	Contains a 5250 data stream that results from a 5250 Read Screen with Extended Attributes command
3270	Contains a 3270 data stream
*FORMAT	Contains a formatted buffer for the data. See the *SNA Upline Facility Programmer's Guide* to determine the format of the buffer header.

See Figure 8-3 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 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
I	5250RS	3270RB
I	5250RSE	3270RB

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.
X'D030'	A data stream resulting from a Read Screen with Extended Attributes command was specified for a display device that does not support extended attributes.

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.

CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 3. Configuration APIs

Chapter 9. Configuration APIs

The configuration application programming interfaces (APIs) allow you to get information about the configuration status of the system and about individual configuration descriptions. The configuration APIs available to you are:

List Configuration Descriptions (QDCLCFGD) returns a list of configuration descriptions.
Retrieve Configuration Status (QDCRCFGS) retrieves the current status of a controller, device, line, or network interface.
Retrieve Controller Description (QDCRCTLD) retrieves information about a controller description.
Retrieve Device Description (QDCRDEVD) retrieves information about a device description.
Retrieve Line Description (QDCRLIND) retrieves information about a line description.

The following sections describe each API in detail. The API descriptions are presented in alphabetical order.

List Configuration Descriptions (QDCLCFGD) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	CHAR(8)
3	Configuration description type	Input	Char(10)
4	Object qualifier	Input	Char(40)
5	Status qualifier	Input	Char(20)
6	Error code	I/O	Char(*)

The List Configuration Descriptions (QDCLCFGD) API returns a list of configuration descriptions. The list elements are all of one type (such as line, controller, or device). The list contents can be further restricted by one or more qualifiers, specified as parameters.

Authorities and Locks

Configuration Description Authority *USE

(If one or more listed objects do not meet the authority requirement, they will be omitted from the returned list. Only objects to which the user has proper authority are in the list.)

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The user space that receives the information, 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 content and format of the list returned. The possible format names are:

CFGD0100 List of selected configuration descriptions
CFGD0200 List of selected configuration descriptions, plus current status of each one

See "Format of Configuration Lists" on page 9-2 for a description of these formats.

Configuration description type
INPUT; CHAR(10)
The type of configuration descriptions to be included in the list. The possible description types are:

***LIND** Line descriptions
***CTLD** Controller descriptions
***DEVD** Device descriptions
***NWID** Network interface descriptions

Object qualifier
INPUT; CHAR(40)
A restriction on the objects to be listed. If a qualifier is specified that is inconsistent with the configuration description type parameter, an error message is returned. A null list is returned if no configuration descriptions meet the qualifications. If this parameter is set to all blanks, no object qualification is performed. This parameter is divided into four 10-character fields.

The primary qualifier value is placed in the first 10 characters of the parameter. The allowable values are:

Object name
Object of this name only
Generic object name
All objects that have names matching the generic string.
***ALL** All lines, controllers or devices
***CMN** Communications controllers or devices only
***LOC** Devices that match the remote location name parameter only
***RSRC** Devices that match the resource name parameter only
***DKT** Diskette devices only
***DSP** Display devices only
***LCLDSP** Local display devices only
***RMTDSP** Remote display devices only
***VRTDSP** Virtual display devices only
***PRT** Printer devices only
***LCLPRT** Local printer devices only

	***RMTPRT**	Remote printer devices only
	***VRTPRT**	Virtual printer devices only
	***SNPT**	SNPT or SNPT-class devices only
	***TAP**	Tape devices only
	***WS**	Work station controllers only
	***LWS**	Local work station controllers only
	***RWS**	Remote work station controllers only
	***VWS**	Virtual work station controllers only
	***ASC**	Asynchronous lines only
	***BSC**	BSC lines only
	***DDI**	Distributed data interface lines only
	***ETH**	Ethernet lines only
	***FAX**	Fax lines only
	***FR**	Frame relay lines only
	***SDLC**	SDLC lines only
	***TDLC**	TDLC lines only
	***TRN**	Token-ring lines only
	***X25**	X.25 lines only

If the configuration description type parameter is *DEVD and the primary qualifier is *LOC, the remote location name to be used is placed in the second 10 characters of the parameter. If a remote location name is specified with any other combination of the configuration description type parameter and primary qualifier, an error message is returned.

If the primary qualifier is *RSRC, the resource name to be used is placed in the second 10 characters of the parameter. If a resource name is specified with any other primary qualifier, an error message is returned.

If the configuration description type parameter is *CTLD or *DEVD, a type qualifier value may be placed in the third 10 characters of the parameter. If a type qualifier is specified with any other value for the configuration description type parameter, an error message is returned.

A list of valid type values can be found in the *Device Configuration Guide*. If an invalid type value is coded, a null list is returned.

If the configuration description type parameter is *CTLD or *DEVD and a type qualifier is coded in the third 10 characters of the parameter, a model qualifier value may be placed in the fourth 10 characters of the parameter. If a model qualifier value is specified with any other value for the configuration description type parameter, or if the type qualifier is blank, an error message is returned.

A list of valid model values can be found in the *Device Configuration Guide*. If an invalid model value is coded, a null list is returned.

Status qualifier

INPUT; CHAR(20)

Specifies a logical operator and a status value that are used to qualify which configuration descriptions are included in the list. The first 10 bytes contain a logical operator, left-justified. The valid values for the logical operator are:

	***GT**	Greater than
	***GE**	Greater than, or equal to
	***LT**	Less than
	***LE**	Less than, or equal to
	***EQ**	Equal to
	***NE**	Not equal to

The second 10 bytes contain a value denoting status, left-justified. The allowed status values, in order of precedence, are:

	***ACTIVE**	Object is active.
	***VARYON**	Object is varied on.
	***VARYOFF**	Object is varied off.

The value in the second 10 bytes has an inherent hierarchy: *ACTIVE is "greater than" *VARYON, and *VARYON is "greater than" *VARYOFF. The two values are used together to form a logical qualifier. For example, "`*GE *VARYON `" causes only objects that are active or varied on to be listed. Objects that are currently varied off are excluded. Both values must be present if either is present. An invalid logical qualifier results in a null list being 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.

Format of Configuration Lists

To request a list of configuration descriptions of a specific type, use format CFGD0100. To request the current status for each description returned, use format CFGD0200.

The configuration description list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section:
  - CFGD0100 format
  - CFGD0200 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.

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."

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format name specified
28	1C	CHAR(10)	Configuration description type specified
38	26	CHAR(40)	Object qualifier specified
78	4E	CHAR(20)	Status qualifier specified
98	62	CHAR(2)	Reserved

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Configuration description type used
10	A	CHAR(40)	Object qualifier used
50	32	CHAR(20)	Status qualifier used
70	46	CHAR(2)	Reserved

CFGD0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Configuration description name
10	A	CHAR(10)	Configuration description category

CFGD0200 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Current status: numeric code
4	4	CHAR(10)	Configuration description name
14	E	CHAR(10)	Configuration description category
24	18	CHAR(20)	Current status: displayable text
44	2C	CHAR(50)	Text description
94	5E	CHAR(10)	Job name
104	68	CHAR(10)	User name
114	72	CHAR(6)	Job number
120	78	CHAR(10)	Pass-through device

Field Descriptions

Configuration description name. The name of an object selected for inclusion in the list.

Configuration description category. The value returned in this field depends on the value specified for the configuration description type parameter when the API was called, as follows:

- If the configuration description type parameter is *LIND (line description), the value is one of the following:

  *ASC
  *BSC
  *DDI
  *ETH
  *FAX
  *FR
  *IDLC
  *NET
  *SDLC
  *TDLC
  *TRN
  *X25

- If the configuration description type parameter is *CTLD (controller description), the value is one of the following:

  *APPC
  *ASC
  *BSC
  *FNC
  *HOST
  *LWS
  *NET
  *RTL
  *RWS
  *TAP
  *VWS
  A specific controller type value

- If the configuration description type parameter is *DEVD (device description), the value is one of the following:

  *APPC
  *ASC
  *BSC
  *DKT
  *DSP
  *FNC
  *HOST
  *INTR
  *NET
  *PRT
  *RTL
  *SNPT
  *SNUF
  *TAP
  A specific device type value

- If the configuration description type parameter is *NWID, the value is returned as a string of blank characters.

The category values are derived from the command used to create the configuration description.

Configuration description type specified. The value specified for the type of configuration description to be included in the list. The possible types are:

**LIND* Line descriptions
**CTLD* Controller descriptions
**DEVD* Device descriptions
**NWID* Network interface descriptions

Configuration description type used. The type of configuration description included in the list.

Current status. The current status of the selected object using two fields:

- Current status: numeric code
- Current status: displayable text

Note: The displayable text will be shown translated in the primary language of the system. The value is one of the following:

Status Numeric Code (decimal)	Status Displayable Text
0	VARIED OFF
10	VARY OFF PENDING
20	VARY ON PENDING
30	VARIED ON
40	CONNECT PENDING
50	SIGNON DISPLAY
60	ACTIVE
63	ACTIVE READER
66	ACTIVE WRITER
70	HELD
75	POWERED OFF
80	RCYPND
90	RCYCNL
95	SYSTEM REQUEST
100	FAILED
103	FAILED READER
106	FAILED WRITER
110	DIAGNOSTIC MODE
111	DAMAGED
112	LOCKED
113	UNKNOWN

Format name specified. The name specified for the format to be used in generating the list. The possible formats are:

CFGD0100 List of selected configuration descriptions
CFGD0200 List of selected configuration descriptions, plus current status of each one

Job name. The name of the job associated with an active device, if applicable.

Job number. The job number portion of a fully qualified job name.

Object qualifier specified. The qualifier values specified that define which objects are to be included in the generated list. See the object qualifier parameter in the "Required Parameter Group" on page 9-1 for details on the possible values.

Object qualifier used. The qualifier values used to determine which objects are included in the list.

Pass-through device. The name of an upstream device used to complete a pass-through session, if applicable.

Reserved. Space included for alignment.

Status qualifier specified. The status values specified that define which configuration descriptions are to be included in the generated list. See the status qualifier parameter in the "Required Parameter Group" on page 9-1 for details on the possible values.

Status qualifier used. The status values used to determine which configuration descriptions are included in the list.

Text description. The text that describes the selected object.

User name. The user name portion of a fully qualified job name.

User space library name specified. The name specified for the library that contains the user space to receive the generated list.

User space name specified. The name specified for the user space that is to receive the generated list.

Error Messages

CPF24B4 E Severe error addressing parameter list.
CPF26A8 E Configuration description type not valid.
CPF26A9 E Object qualifier not valid.
CPF26AA E Status qualifier not valid.
CPF3C21 E Format name &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Configuration Status (QDCRCFGS) 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	Configuration description type	Input	Char(10)
5	Configuration description name	Input	Char(10)
6	Error code	I/O	Char(*)

The Retrieve Configuration Status (QDCRCFGS) API retrieves the current status of a line, controller, device, or network interface.

Authorities and Locks

Configuration Description Authority *USE

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the status information. Format CFGS0100 describes the layout of the status information returned in this variable.

Length of receiver variable
INPUT; BINARY(4)
The length of the area referenced by the receiver variable parameter. If the amount of information to be returned is greater than this value, the information is truncated to this length.

Format name
INPUT; CHAR(8)
The content and format of the status information returned for the specified configuration description. The format name is:

CFGS0100 Configuration description object status

See "Format of Status Information" for a description of this format.

Configuration description type
INPUT; CHAR(10)
The type of configuration description object for which status is being retrieved. The possible values are:

***LIND** The object is a line description.
***CTLD** The object is a controller description.
***DEVD** The object is a device description.
***NWID** The object is a network interface description.

Configuration description name
INPUT; CHAR(10)
The name of the line, controller, device or network interface for which status is being retrieved.

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 Status Information

Following is the format of the status information returned. For detailed descriptions of the fields in the list, see "Field Descriptions."

CFGS0100 Format

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Current status: numeric code
12	C	CHAR(7)	Date information retrieved
19	13	CHAR(6)	Time information retrieved
25	19	CHAR(20)	Current Status: displayable text
45	2D	CHAR(10)	Job name
55	37	CHAR(10)	User name
65	41	CHAR(6)	Job number
71	47	CHAR(10)	Pass-through device

Field Descriptions

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.

Current status. The value will be one of the following:

Status Numeric Code (decimal)	Status Displayable Text
0	VARIED OFF
10	VARY OFF PENDING
20	VARY ON PENDING
30	VARIED ON
40	CONNECT PENDING
50	SIGNON DISPLAY
60	ACTIVE
63	ACTIVE READER
66	ACTIVE WRITER
70	HELD
75	POWERED OFF
80	RCYPND
90	RCYCNL
95	SYSTEM REQUEST
100	FAILED
103	FAILED READER
106	FAILED WRITER
110	DIAGNOSTIC MODE
111	DAMAGED
112	LOCKED
113	UNKNOWN

Date information retrieved. The date that the information was provided by the API. This is returned as 7 characters in the form CYYMMDD, where:

C Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
YY Year

MM Month
DD Day

Job name. The name of the job associated with an active device, if applicable.

Job number. The job number portion of a fully qualified job name.

Pass-through device. The name of an upstream device used to complete a pass-through session, if applicable.

Time information retrieved. The time that the information was provided by the API. It is returned as 6 characters in the form HHMMSS, where:

HH Hour
MM Minute
SS Second

User name. The user name portion of a fully qualified job name.

Error Messages

CPF24B4 E Severe error addressing parameter list.
CPF2625 E Not able to allocate object &1.
CPF2634 E Not authorized to object &1.
CPF26A8 E Configuration description type not valid.
CPF2702 E Device description &1 not found.
CPF2703 E Controller description &1 not found.
CPF2704 E Line description &1 not found.
CPF27A4 E NWI description &1 not found.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 not valid.
CPF3C24 E Length of receiver variable not valid.
CPF3CF1 E Error code parameter not valid.
CPF8104 E Controller description &4 damaged.
CPF8105 E Device description &4 damaged.
CPF811D E NWI description &4 damaged.
CPF8125 E Line description &4 damaged.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Controller Description (QDCRCTLD) 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	Controller name	Input	Char(10)
5	Error code	I/O	Char(*)

The Retrieve Controller Description (QDCRCTLD) API retrieves information about a controller description.

Authorities and Locks

Controller Description Authority *USE
Device Description Authority *USE
Controller Description Lock *EXCLRD
Device Description Lock *EXCLRD

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the controller information.

Length of receiver variable
INPUT; BINARY(4)
The length of the area referenced by the receiver variable parameter. If the amount of information to be returned is greater than this value, the information will be truncated to this length.

Format name
INPUT; CHAR(8)
The content and format of the information returned for each controller description. The possible format names are:

CTLD0100 Basic controller information.
CTLD0200 Basic controller information, plus list of attached devices.
CTLD0300 Detailed information for controller category *APPC
CTLD0400 Detailed information for controller category *ASC
CTLD0500 Detailed information for controller category *BSC
CTLD0600 Detailed information for controller category *FNC
CTLD0700 Detailed information for controller category *HOST
CTLD0800 Detailed information for controller category *NET
CTLD0900 Detailed information for controller category *RTL
CTLD1000 Detailed information for controller category *RWS
CTLD1100 Detailed information for controller category *VWS
CTLD1200 Detailed information for controller category *LWS
CTLD1300 Detailed information for controller category *TAP

See "Format of Controller Information" on page 9-7 for a description of these formats.

Controller name
INPUT; CHAR(10)
The name of the controller description to be retrieved.

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 Controller Information

When the controller category is unknown, specify CTLD0100 or CTLD0200 and the basic information (including controller category) will be returned. When the controller category is known, specify one of the other category-specific formats.

For detailed descriptions of the fields returned in these formats, see "Field Descriptions" on page 9-16.

CTLD0100 Format: Use this format to find out the controller category, plus some very basic information about the controller. Then you may use the returned controller category to select one of the other (category-specific) formats to call the API again for detailed information about the controller description. This format also returns the number of devices currently attached to this controller.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of attached devices
12	C	CHAR(7)	Date information retrieved
19	13	CHAR(6)	Time information retrieved
25	19	CHAR(10)	Controller name
35	23	CHAR(10)	Controller category
45	2D	CHAR(10)	Online at IPL
55	37	CHAR(50)	Text description
105	69	CHAR(3)	Reserved

CTLD0200 Format: This format returns basic controller information, plus a list of attached devices. Some basic information is also included for each attached device.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Offset to list of attached devices.
112	70	BINARY(4)	Entry length for list of attached devices
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(10)	Device category
		CHAR(10)	Device type
		CHAR(50)	Device text description

CTLD0300 Format: This format returns detailed information about a controller of category *APPC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Maximum frame size
112	70	BINARY(4)	IDLC default window size
116	74	BINARY(4)	IDLC frame retry
120	78	BINARY(4)	IDLC response timer
124	7C	BINARY(4)	IDLC connect retry
128	80	BINARY(4)	Predial delay
132	84	BINARY(4)	Redial delay
136	88	BINARY(4)	Dial retries
140	8C	BINARY(4)	Disconnect timer: minimum connect
144	90	BINARY(4)	Disconnect timer: disconnect delay
148	94	BINARY(4)	Short-hold mode disconnect limit
152	98	BINARY(4)	Short-hold mode disconnect timer
156	9C	BINARY(4)	SDLC poll limit
160	A0	BINARY(4)	SDLC out limit
164	A4	BINARY(4)	SDLC connect poll retry
168	A8	BINARY(4)	SDLC NDM poll timer
172	AC	BINARY(4)	LAN frame retry
176	B0	BINARY(4)	LAN connection retry
180	B4	BINARY(4)	LAN response timer
184	B8	BINARY(4)	LAN connection timer
188	BC	BINARY(4)	LAN acknowledgement timer
192	C0	BINARY(4)	LAN inactivity timer
196	C4	BINARY(4)	LAN acknowledgement frequency
200	C8	BINARY(4)	LAN maximum outstanding frames
204	CC	BINARY(4)	LAN access priority
208	D0	BINARY(4)	LAN window step
212	D4	BINARY(4)	Default packet size: transmit

Offset		Type	Field
Dec	Hex		
216	D8	BINARY(4)	Default packet size: receive
220	DC	BINARY(4)	Negotiated packet size: transmit
224	E0	BINARY(4)	Negotiated packet size: receive
228	E4	BINARY(4)	Default window size: transmit
232	E8	BINARY(4)	Default window size: receive
236	EC	BINARY(4)	Negotiated window size: transmit
240	F0	BINARY(4)	Negotiated window size: receive
244	F4	BINARY(4)	X.25 frame retry
248	F8	BINARY(4)	X.25 connection retry
252	FC	BINARY(4)	X.25 response timer
256	100	BINARY(4)	X.25 connection timer
260	104	BINARY(4)	X.25 delayed connection timer
264	108	BINARY(4)	X.25 acknowledgement timer
268	10C	BINARY(4)	X.25 inactivity timer
272	110	BINARY(4)	APPN transmission group number
276	114	BINARY(4)	Autodelete device
280	118	BINARY(4)	User-defined 1
284	11C	BINARY(4)	User-defined 2
288	120	BINARY(4)	User-defined 3
292	124	BINARY(4)	Recovery limits: count limit
296	128	BINARY(4)	Recovery limits: time interval
300	12C	BINARY(4)	Offset to list of attached devices
304	130	BINARY(4)	Entry length for list of attached devices
308	134	BINARY(4)	Offset to list of switched lines
312	138	BINARY(4)	Number of switched lines
316	13C	BINARY(4)	Entry length for list of switched lines
320	140	CHAR(10)	Link type
330	14A	CHAR(10)	Controller type
340	154	CHAR(10)	Switched connection
350	15E	CHAR(10)	Short-hold mode
360	168	CHAR(10)	Switched network backup
370	172	CHAR(10)	Activate switched network backup
380	17C	CHAR(10)	APPN capable
390	186	CHAR(10)	Attached nonswitched line name
400	190	CHAR(10)	Character code
410	19A	CHAR(10)	Remote network identifier
420	1A4	CHAR(10)	Remote control point name
430	1AE	CHAR(10)	Exchange identifier
440	1B8	CHAR(12)	System service control point identifier
452	1C4	CHAR(10)	Initial connection
462	1CE	CHAR(10)	Dial initiation
472	1D8	CHAR(32)	Connection number
504	1F8	CHAR(10)	Answer number
514	202	CHAR(10)	Activate X.25 network address
524	20C	CHAR(10)	Connection list
534	216	CHAR(10)	Connection list entry
544	220	CHAR(10)	Switched disconnect
554	22A	CHAR(10)	Data link role
564	234	CHAR(10)	Station address
574	23E	CHAR(10)	SDLC poll priority
584	248	CHAR(12)	LAN remote adapter address
596	254	CHAR(10)	Destination service access point
606	25E	CHAR(10)	Source service access point
616	268	CHAR(10)	X.25 network level
626	272	CHAR(10)	X.25 link protocol
636	27C	CHAR(10)	X.25 logical channel ID
646	286	CHAR(10)	X.25 connection password
656	290	CHAR(10)	X.25 switched line selection
666	29A	CHAR(10)	X.25 user group ID
676	2A4	CHAR(10)	X.25 reverse charging
686	2AE	CHAR(10)	APPC CP session support
696	2B8	CHAR(10)	APPN node type
706	2C2	CHAR(10)	APPN minimum switched status
716	2CC	CHAR(10)	Model controller description
726	2D6	CHAR(10)	Connection network identifier
736	2E0	CHAR(10)	Connection network CP name

Offset		Type	Field
Dec	Hex		
746	2EA	CHAR(10)	Control owner
756	2F4	CHAR(218)	User facilities
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved
These fields repeat for each switched line		CHAR(10)	Switched line name
		CHAR(2)	Reserved

CTLD0400 Format: This format returns detailed information about a controller of category *ASC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Predial delay
112	70	BINARY(4)	Redial delay
116	74	BINARY(4)	Dial retries
120	78	BINARY(4)	File transfer acknowledgement timer
124	7C	BINARY(4)	File transfer retry
128	80	BINARY(4)	Default packet size: transmit
132	84	BINARY(4)	Default packet size: receive
136	88	BINARY(4)	Negotiated packet size: transmit
140	8C	BINARY(4)	Negotiated packet size: receive
144	90	BINARY(4)	Default window size: transmit
148	94	BINARY(4)	Default window size: receive
152	98	BINARY(4)	Negotiated window size: transmit
156	9C	BINARY(4)	Negotiated window size: receive
160	A0	BINARY(4)	Recovery limits: count limit
164	A4	BINARY(4)	Recovery limits: time interval
168	A8	BINARY(4)	Offset to list of attached devices
172	AC	BINARY(4)	Entry length for list of attached devices
176	B0	BINARY(4)	Offset to list of switched lines
180	B4	BINARY(4)	Number of entries in list of switched lines
184	B8	BINARY(4)	Entry length for list of switched lines
188	BC	CHAR(10)	Link type
198	C6	CHAR(10)	Switched connection
208	D0	CHAR(10)	Switched network backup
218	DA	CHAR(10)	Activate switched network backup
228	E4	CHAR(10)	Attached nonswitched line name
238	EE	CHAR(10)	Initial connection
248	F8	CHAR(32)	Connection number
280	118	CHAR(10)	Answer number
290	122	CHAR(10)	Activate X.25 network address
300	12C	CHAR(10)	Switched disconnect
310	136	CHAR(10)	Remote verify
320	140	CHAR(10)	Local location name
330	14A	CHAR(10)	Local identifier
340	154	CHAR(10)	PAD emulation
350	15E	CHAR(10)	X.25 logical channel ID
360	168	CHAR(10)	X.25 switched line selection
370	172	CHAR(10)	X.25 user group ID
380	17C	CHAR(10)	X.25 reverse charging
390	186	CHAR(218)	User facilities
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved
These fields repeat for each switched line		CHAR(10)	Switched line name
		CHAR(2)	Reserved

CTLD0500 Format: This format returns detailed information about a controller of category *BSC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Predial delay
112	70	BINARY(4)	Redial delay
116	74	BINARY(4)	Dial retries
120	78	BINARY(4)	Recovery limits: count limit

Offset		Type	Field
Dec	Hex		
124	7C	BINARY(4)	Recovery limits: time interval
128	80	BINARY(4)	Offset to list of attached devices
132	84	BINARY(4)	Entry length for list of attached devices
136	88	BINARY(4)	Offset to list of switched lines
140	8C	BINARY(4)	Number of entries in list of switched lines
144	90	BINARY(4)	Entry length for list of switched lines
148	94	BINARY(4)	Offset to list of remote identifiers
152	98	BINARY(4)	Number of entries in list of remote identifiers
156	9C	BINARY(4)	Entry length for list of remote identifiers
160	A0	CHAR(10)	Connection type
170	AA	CHAR(10)	Switched network backup
180	B4	CHAR(10)	Activate switched network backup
190	BE	CHAR(10)	Attached nonswitched line name
200	C8	CHAR(10)	Application type
210	D2	CHAR(10)	Initial connection
220	DC	CHAR(32)	Connection number
252	FC	CHAR(10)	Local identifier
262	106	CHAR(10)	RJE host type
272	110	CHAR(80)	RJE host signon/logon
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved
These fields repeat for each switched line		CHAR(10)	Switched line name
		CHAR(2)	Reserved
These fields repeat for each remote identifier		CHAR(30)	Remote identifier
		CHAR(2)	Reserved

CTLD0600 Format: This format returns detailed information about a controller of category *FNC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Maximum frame size
112	70	BINARY(4)	Predial delay
116	74	BINARY(4)	Redial delay
120	78	BINARY(4)	Dial retries
124	7C	BINARY(4)	Short-hold mode disconnect limit
128	80	BINARY(4)	Short-hold mode disconnect timer
132	84	BINARY(4)	SDLC poll limit
136	88	BINARY(4)	SDLC out limit
140	8C	BINARY(4)	SDLC connect poll retry
144	90	BINARY(4)	SDLC NDM poll timer
148	94	BINARY(4)	LAN frame retry
152	98	BINARY(4)	LAN connection retry
156	9C	BINARY(4)	LAN response timer
160	A0	BINARY(4)	LAN connection timer
164	A4	BINARY(4)	LAN acknowledgement timer
168	A8	BINARY(4)	LAN inactivity timer
172	AC	BINARY(4)	LAN acknowledgement frequency
176	B0	BINARY(4)	LAN maximum outstanding frames
180	B4	BINARY(4)	LAN access priority
184	B8	BINARY(4)	LAN window step
188	BC	BINARY(4)	Default packet size: transmit
192	C0	BINARY(4)	Default packet size: receive
196	C4	BINARY(4)	Negotiated packet size: transmit
200	C8	BINARY(4)	Negotiated packet size: receive
204	CC	BINARY(4)	Default window size: transmit
208	D0	BINARY(4)	Default window size: receive
212	D4	BINARY(4)	Negotiated window size: transmit
216	D8	BINARY(4)	Negotiated window size: receive
220	DC	BINARY(4)	X.25 frame retry
224	E0	BINARY(4)	X.25 connection retry
228	E4	BINARY(4)	X.25 response timer
232	E8	BINARY(4)	X.25 connection timer
236	EC	BINARY(4)	X.25 delayed connection timer
240	F0	BINARY(4)	X.25 acknowledgement timer

Offset			
Dec	Hex	Type	Field
244	F4	BINARY(4)	X.25 inactivity timer
248	F8	BINARY(4)	Recovery limits: count limit
252	FC	BINARY(4)	Recovery limits: time interval
256	100	BINARY(4)	Offset to list of attached devices
260	104	BINARY(4)	Entry length for list of attached devices
264	108	BINARY(4)	Offset to list of switched lines
268	10C	BINARY(4)	Number of entries in list of switched lines
272	110	BINARY(4)	Entry length for list of switched lines
276	114	CHAR(10)	Controller type
286	11E	CHAR(10)	Controller model
296	128	CHAR(10)	Link type
306	132	CHAR(10)	Switched connection
316	13C	CHAR(10)	Short-hold mode
326	146	CHAR(10)	Switched network backup
336	150	CHAR(10)	Activate switched network backup
346	15A	CHAR(10)	Attached nonswitched line name
356	164	CHAR(10)	Character code
366	16E	CHAR(10)	Exchange identifier
376	178	CHAR(12)	System service control point identifier
388	184	CHAR(10)	Initial connection
398	18E	CHAR(32)	Connection number
430	1AE	CHAR(10)	Answer number
440	1B8	CHAR(10)	Activate X.25 network address
450	1C2	CHAR(10)	Switched disconnect
460	1CC	CHAR(10)	Station address
470	1D6	CHAR(10)	SDLC poll priority
480	1E0	CHAR(12)	LAN remote adapter address
492	1EC	CHAR(10)	Destination service access point
502	1F6	CHAR(10)	Source service access point
512	200	CHAR(10)	X.25 network level
522	20A	CHAR(10)	X.25 link protocol
532	214	CHAR(10)	X.25 logical channel ID
542	21E	CHAR(10)	X.25 connection password
552	228	CHAR(10)	X.25 switched line selection
562	232	CHAR(10)	X.25 user group ID
572	23C	CHAR(10)	X.25 reverse charging
582	246	CHAR(218)	User facilities
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved
These fields repeat for each switched line		CHAR(10)	Switched line name
		CHAR(2)	Reserved

CTLD0700 Format: This format returns detailed information about a controller of category *HOST.

Offset			
Dec	Hex	Type	Field
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Maximum frame size
112	70	BINARY(4)	IDLC default window size
116	74	BINARY(4)	IDLC frame retry
120	78	BINARY(4)	IDLC response timer
124	7C	BINARY(4)	IDLC connect retry
128	80	BINARY(4)	Predial delay
132	84	BINARY(4)	Redial delay
136	88	BINARY(4)	Dial retries
140	8C	BINARY(4)	Disconnect timer: minimum connect
144	90	BINARY(4)	Disconnect timer: disconnect delay
148	94	BINARY(4)	LAN frame retry
152	98	BINARY(4)	LAN connection retry
156	9C	BINARY(4)	LAN response timer
160	A0	BINARY(4)	LAN connection timer
164	A4	BINARY(4)	LAN acknowledgement timer
168	A8	BINARY(4)	LAN inactivity timer
172	AC	BINARY(4)	LAN acknowledgement frequency
176	B0	BINARY(4)	LAN maximum outstanding frames
180	B4	BINARY(4)	LAN access priority
184	B8	BINARY(4)	LAN window step
188	BC	BINARY(4)	Default packet size: transmit

Offset		Type	Field
Dec	Hex		
192	C0	BINARY(4)	Default packet size: receive
196	C4	BINARY(4)	Negotiated packet size: transmit
200	C8	BINARY(4)	Negotiated packet size: receive
204	CC	BINARY(4)	Default window size: transmit
208	D0	BINARY(4)	Default window size: receive
212	D4	BINARY(4)	Negotiated window size: transmit
216	D8	BINARY(4)	Negotiated window size: receive
220	DC	BINARY(4)	X.25 frame retry
224	E0	BINARY(4)	X.25 response timer
228	E4	BINARY(4)	X.25 acknowledgement timer
232	E8	BINARY(4)	X.25 inactivity timer
236	EC	BINARY(4)	APPN transmission group number
240	F0	BINARY(4)	Autodelete device
244	F4	BINARY(4)	User-defined 1
248	F8	BINARY(4)	User-defined 2
252	FC	BINARY(4)	User-defined 3
256	100	BINARY(4)	Recovery limits: count limit
260	104	BINARY(4)	Recovery limits: time interval
264	108	BINARY(4)	Offset to list of attached devices
268	10C	BINARY(4)	Entry length for list of attached devices
272	110	BINARY(4)	Offset to list of switched lines
276	114	BINARY(4)	Number of entries in list of switched lines
280	118	BINARY(4)	Entry length for list of switched lines
284	11C	CHAR(10)	Link type
294	126	CHAR(10)	Switched connection
304	130	CHAR(10)	Short-hold mode
314	13A	CHAR(10)	Switched network backup
324	144	CHAR(10)	Activate switched network backup
334	14E	CHAR(10)	APPN capable
344	158	CHAR(10)	Attached nonswitched line name
354	162	CHAR(10)	Character code
364	16C	CHAR(10)	Remote network identifier
374	176	CHAR(10)	Remote control point name
384	180	CHAR(10)	Adjacent link station
394	18A	CHAR(12)	System service control point identifier
406	196	CHAR(10)	Local exchange identifier
416	1A0	CHAR(10)	Initial connection
426	1AA	CHAR(10)	Dial initiation
436	1B4	CHAR(32)	Connection number
468	1D4	CHAR(10)	Answer number
478	1DE	CHAR(10)	Activate X.25 network address
488	1E8	CHAR(10)	Connection list
498	1F2	CHAR(10)	Connection list entry
508	1FC	CHAR(10)	Switched disconnect
518	206	CHAR(10)	Station address
528	210	CHAR(12)	LAN remote adapter address
540	21C	CHAR(10)	Destination service access point
550	226	CHAR(10)	Source service access point
560	230	CHAR(10)	X.25 network level
570	23A	CHAR(10)	X.25 link protocol
580	244	CHAR(10)	X.25 logical channel ID
590	24E	CHAR(10)	X.25 connection password
600	258	CHAR(10)	X.25 switched line selection
610	262	CHAR(10)	X.25 user group ID
620	26C	CHAR(10)	X.25 reverse charging
630	276	CHAR(10)	APPC CP session support
640	280	CHAR(10)	APPN node type
650	28A	CHAR(10)	APPN minimum switched status
660	294	CHAR(10)	Recontact at vary off
670	29E	CHAR(10)	Autocreate device
680	2A8	CHAR(218)	User facilities
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved
These fields repeat for each switched line		CHAR(10)	Switched line name
		CHAR(2)	Reserved

CTLD0800 Format: This format returns detailed information about a controller of category *NET.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Connection response timer
112	70	BINARY(4)	Offset to list of attached devices
116	74	BINARY(4)	Entry length for list of attached devices
120	78	CHAR(10)	Attached line
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved

CTLD0900 Format: This format returns detailed information about a controller of category *RTL.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Maximum frame size
112	70	BINARY(4)	Predial delay
116	74	BINARY(4)	Redial delay
120	78	BINARY(4)	Dial retries
124	7C	BINARY(4)	SDLC poll limit
128	80	BINARY(4)	SDLC out limit
132	84	BINARY(4)	SDLC connect poll retry
136	88	BINARY(4)	SDLC NDM poll timer
140	8C	BINARY(4)	LAN frame retry
144	90	BINARY(4)	LAN connection retry
148	94	BINARY(4)	LAN response timer
152	98	BINARY(4)	LAN connection timer
156	9C	BINARY(4)	LAN acknowledgement timer
160	A0	BINARY(4)	LAN inactivity timer
164	A4	BINARY(4)	LAN acknowledgement frequency
168	A8	BINARY(4)	LAN maximum outstanding frames
172	AC	BINARY(4)	LAN access priority
176	B0	BINARY(4)	LAN window step
180	B4	BINARY(4)	Default packet size: transmit
184	B8	BINARY(4)	Default packet size: receive
188	BC	BINARY(4)	Negotiated packet size: transmit
192	C0	BINARY(4)	Negotiated packet size: receive
196	C4	BINARY(4)	Default window size: transmit
200	C8	BINARY(4)	Default window size: receive
204	CC	BINARY(4)	Negotiated window size: transmit
208	D0	BINARY(4)	Negotiated window size: receive
212	D4	BINARY(4)	X.25 frame retry
216	D8	BINARY(4)	X.25 connection retry
220	DC	BINARY(4)	X.25 response timer
224	E0	BINARY(4)	X.25 connection timer
228	E4	BINARY(4)	X.25 delayed connection timer
232	E8	BINARY(4)	Recovery limits: count limit
236	EC	BINARY(4)	Recovery limits: time interval
240	F0	BINARY(4)	Offset to list of attached devices
244	F4	BINARY(4)	Entry length for list of attached devices
248	F8	BINARY(4)	Offset to list of switched lines
252	FC	BINARY(4)	Number of entries in list of switched lines
256	100	BINARY(4)	Entry length for list of switched lines
260	104	CHAR(10)	Controller type
270	10E	CHAR(10)	Controller model
280	118	CHAR(10)	Link type
290	122	CHAR(10)	Switched line
300	12C	CHAR(10)	Switched network backup
310	136	CHAR(10)	Activate switched network backup
320	140	CHAR(10)	Attached nonswitched line name
330	14A	CHAR(10)	Character code
340	154	CHAR(10)	Exchange identifier
350	15E	CHAR(12)	System service control point identifier
362	16A	CHAR(10)	Initial connection
372	174	CHAR(32)	Connection number
404	194	CHAR(10)	Answer number

Offset		Type	Field
Dec	Hex		
414	19E	CHAR(10)	Activate X.25 network address
424	1A8	CHAR(10)	Switched disconnect
434	1B2	CHAR(10)	Station address
444	1BC	CHAR(10)	SDLC poll priority
454	1C6	CHAR(12)	LAN remote adapter address
466	1D2	CHAR(10)	Destination service access point
476	1DC	CHAR(10)	Source service access point
486	1E6	CHAR(10)	X.25 network level
496	1F0	CHAR(10)	X.25 logical channel ID
506	1FA	CHAR(10)	X.25 connection password
516	204	CHAR(10)	X.25 switched line selection
526	20E	CHAR(10)	X.25 user group ID
536	218	CHAR(10)	X.25 reverse charging
546	222	CHAR(218)	User facilities
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved
These fields repeat for each switched line		CHAR(10)	Switched line name
		CHAR(2)	Reserved

CTLD1000 Format: This format returns detailed information about a controller of category *RWS.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Device wait timer
112	70	BINARY(4)	Maximum frame size
116	74	BINARY(4)	IDLC default window size
120	78	BINARY(4)	IDLC frame retry
124	7C	BINARY(4)	IDLC response timer
128	80	BINARY(4)	IDLC connect retry
132	84	BINARY(4)	Predial delay
136	88	BINARY(4)	Redial delay
140	8C	BINARY(4)	Dial retries
144	90	BINARY(4)	Short-hold mode disconnect limit
148	94	BINARY(4)	Short-hold mode disconnect timer
152	98	BINARY(4)	SDLC poll limit
156	9C	BINARY(4)	SDLC out limit
160	A0	BINARY(4)	SDLC connect poll retry
164	A4	BINARY(4)	SDLC NDM poll timer
168	A8	BINARY(4)	LAN frame retry
172	AC	BINARY(4)	LAN connection retry
176	B0	BINARY(4)	LAN response timer
180	B4	BINARY(4)	LAN connection timer
184	B8	BINARY(4)	LAN acknowledgement timer
188	BC	BINARY(4)	LAN inactivity timer
192	C0	BINARY(4)	LAN acknowledgement frequency
196	C4	BINARY(4)	LAN maximum outstanding frames
200	C8	BINARY(4)	LAN access priority
204	CC	BINARY(4)	LAN window step
208	D0	BINARY(4)	Default packet size: transmit
212	D4	BINARY(4)	Default packet size: receive
216	D8	BINARY(4)	Negotiated packet size: transmit
220	DC	BINARY(4)	Negotiated packet size: receive
224	E0	BINARY(4)	Default window size: transmit
228	E4	BINARY(4)	Default window size: receive
232	E8	BINARY(4)	Negotiated window size: transmit
236	EC	BINARY(4)	Negotiated window size: receive
240	F0	BINARY(4)	X.25 frame retry
244	F4	BINARY(4)	X.25 connection retry
248	F8	BINARY(4)	X.25 response timer
252	FC	BINARY(4)	X.25 connection timer
256	100	BINARY(4)	X.25 delayed connection timer
260	104	BINARY(4)	X.25 acknowledgement timer
264	108	BINARY(4)	X.25 inactivity timer
268	10C	BINARY(4)	Allocation retry timer
272	110	BINARY(4)	Recovery limits: count limit
276	114	BINARY(4)	Recovery limits: time interval

Offset			
Dec	Hex	Type	Field
280	118	BINARY(4)	Offset to list of attached devices
284	11C	BINARY(4)	Entry length for list of attached devices
288	120	BINARY(4)	Offset to list of switched lines
292	124	BINARY(4)	Number of entries in list of switched lines
296	128	BINARY(4)	Entry length for list of switched lines
300	12C	CHAR(10)	Controller type
310	136	CHAR(10)	Controller model
320	140	CHAR(10)	Link type
330	14A	CHAR(10)	Switched connection
340	154	CHAR(10)	Short-hold mode
350	15E	CHAR(10)	Switched network backup
360	168	CHAR(10)	Activate switched network backup
370	172	CHAR(10)	Attached nonswitched line name
380	17C	CHAR(10)	TDLC line name
390	186	CHAR(10)	Character code
400	190	CHAR(10)	Remote location name
410	19A	CHAR(10)	Local location name
420	1A4	CHAR(10)	Remote network identifier
430	1AE	CHAR(10)	Exchange identifier
440	1B8	CHAR(12)	System service control point identifier
452	1C4	CHAR(10)	Initial connection
462	1CE	CHAR(10)	Dial initiation
472	1D8	CHAR(32)	Connection number
504	1F8	CHAR(10)	Answer number
514	202	CHAR(10)	Activate X.25 network address
524	20C	CHAR(10)	Connection list
534	216	CHAR(10)	Connection list entry
544	220	CHAR(10)	Station address
554	22A	CHAR(10)	SDLC poll priority
564	234	CHAR(12)	LAN remote adapter address
576	240	CHAR(10)	Destination service access point
586	24A	CHAR(10)	Source service access point
596	254	CHAR(10)	X.25 network level
606	25E	CHAR(10)	X.25 link protocol
616	268	CHAR(10)	X.25 logical channel ID
626	272	CHAR(10)	X.25 connection password
636	27C	CHAR(10)	X.25 switched line selection
646	286	CHAR(10)	X.25 user group ID
656	290	CHAR(10)	X.25 reverse charging
666	29A	CHAR(218)	User facilities
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved
These fields repeat for each switched line		CHAR(10)	Switched line name
		CHAR(2)	Reserved

CTLD1100 Format: This format returns detailed information about a controller of category *VWS.

Offset			
Dec	Hex	Type	Field
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Offset to list of attached devices
112	70	BINARY(4)	Entry length for list of attached devices
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved

CTLD1200 Format: This format returns detailed information about a controller of category *LWS.

Offset			
Dec	Hex	Type	Field
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Device wait timer
112	70	BINARY(4)	Offset to list of attached devices
116	74	BINARY(4)	Entry length for list of attached devices
120	78	CHAR(10)	Controller type
130	82	CHAR(10)	Controller model
140	8C	CHAR(10)	Resource name
150	96	CHAR(10)	TDLC line name
160	A0	CHAR(10)	Automatic configuration

Offset		Type	Field
Dec	Hex		
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved

CTLD1300 Format: This format returns detailed information about a controller of category *TAP.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format CTLD0100
108	6C	BINARY(4)	Offset to list of attached devices
112	70	BINARY(4)	Entry length for list of attached devices
116	74	CHAR(10)	Controller type
126	7E	CHAR(10)	Controller model
136	88	CHAR(10)	Resource name
146	92	CHAR(10)	Automatic configuration
These fields repeat for each attached device		CHAR(10)	Attached device name
		CHAR(2)	Reserved

Field Descriptions

Fields in the various formats returned by this API are described in the chapter on communications controller descriptions in the *OS/400* Communications Configuration Reference*.

In certain cases, numeric values are assigned by this API to represent character values for some of the returned fields. Where a numeric value is assigned, the numeric value and the equivalent character value are listed as an *Exception* in the following field descriptions.

Activate switched network backup (ACTSNBU). Shows whether the switched network backup is activated.

Activate X.25 network address (ACTXADR). The current X.25 network address for active X.25 controller descriptions.

Adjacent link station (ASJLNKSTN). The link station name of the adjacent system. (See the ADJLNKSTN keyword in the *OS/400* Communications Configuration Reference*.)

Allocation retry timer (ALCRTYTMR). The time to wait between attempts to activate devices associated with this controller. (See the ALCRTYTMR keyword in the *OS/400* Communications Configuration Reference*.)

Answer number (ANSNBR). The X.25 network addresses from which this controller can accept calls. (See the ANSNBR keyword in the *OS/400* Communications Configuration Reference*.)

APPC CP session support (CPSSN). Whether this controller supports control point-to-control point sessions. (See the CPSSN keyword in the *OS/400* Communications Configuration Reference*.)

Application type (APPTYPE). The type of application that this controller is to be used for. (See the APPTYPE keyword in the *OS/400* Communications Configuration Reference*.)

APPN capable (APPN). Whether the local AS/400 system appears to the adjacent system as either a network node or an end node in the local system network attributes, or the local system appears to the adjacent system as a low-entry networking node. (See the APPN keyword in the *OS/400* Communications Configuration Reference*.)

APPN minimum switched status (MINSWTSTS). The minimum status required for this controller description to be considered eligible for APPN routing. (See the MINSWTSTS keyword in the *OS/400* Communications Configuration Reference*.)

APPN node type (NODETYPE). The type of node that this controller represents. (See the NODETYPE keyword in the *OS/400* Communications Configuration Reference*.)

APPN transmission group number (TMSGRPNBR). The value to be used by the APPN support for transmission group negotiation with the remote system. (See the TMSGRPNBR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

Attached device name (DEV). The name of one or more devices to be attached to this controller. (See the DEV keyword in the *OS/400* Communications Configuration Reference*.)

Attached line. For network controllers, the name of the line that connects the network to the AS/400 system.

Attached nonswitched line name (LINE). The name of the line description that connects the network to the AS/400 system. (See the LINE keyword in the *OS/400* Communications Configuration Reference*.)

Autocreate device (AUTOCRTDEV). Which devices are automatically created. (See the AUTOCRTDEV keyword in the *OS/400* Communications Configuration Reference*.)

Autodelete device (AUTODLTDEV). The number of minutes an automatically created device can remain in an idle state with no bound sessions and no active conversations on the device. (See the AUTODLTDEV keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -2 implies *NO

Automatic configuration. Whether this controller has been configured automatically.

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 (CODE). The type of character code (EBCDIC or ASCII) used to send the information in a remote work station data stream over the communications line. (See the CODE keyword in the *OS/400* Communications Configuration Reference*.)

Connection list (CNNLSTOUT). The name of a connection list containing the network-assigned numbers used for outgoing calls on this controller. (See the CNNLSTOUT keyword in the *OS/400* Communications Configuration Reference*.)

Connection list entry (CNNLSTOUTE). The name of the connection list entry containing the network-assigned numbers used for outgoing calls on this controller. (See the CNNLSTOUTE keyword in the *OS/400* Communications Configuration Reference*.)

Connection network CP name (CNNCPNAME). The name of the connection network control point. (See the CNNCPNAME keyword in the *OS/400* Communications Configuration Reference*.)

Connection network identifier (CNNNETID). The name of the connection network identifier. (See the CNNNETID keyword in the *OS/400* Communications Configuration Reference*.)

Connection number (CNNNBR). The number (for a switched connection or a nonswitched connection with switched network backup) of the remote controller that is called from the AS/400 system to establish a connection. (See the CNNNBR keyword in the *OS/400* Communications Configuration Reference*.)

Connection response timer (CNNRSPTMR). The amount of time to wait for a response to an incoming connection request. (See the CNNRSPTMR keyword in the *OS/400* Communications Configuration Reference*.)

Connection type (CNN). The type of connection this BSC controller will be used on. (See the CNN keyword in the *OS/400* Communications Configuration Reference*.)

Control owner (CTLOWN). Whether this description is owned by the system or the user. *USER identifies the user as the owner, and *SYS identifies the system as the owner. If the system is the control owner, the user cannot make any changes to the description.

Controller category. This value will be one of the following:

*APPC
*ASC
*BSC
*FNC
*HOST
*LWS
*NET
*RTL
*RWS
*TAP
*VWS

The category value is derived from the command used to create the controller description.

Controller model (MODEL). The model number of the controller. (See the MODEL keyword in the *OS/400* Communications Configuration Reference*.)

Controller name. The name of the controller description.

Controller type (TYPE). The type of controller being described. (See the TYPE keyword in the *OS/400* Communications Configuration Reference*.)

Data link role (ROLE). Whether the remote system is primary, secondary, or dynamically negotiates its role. (See the ROLE keyword in the *OS/400* Communications Configuration Reference*.)

Date information retrieved. The date that the information was provided by the API. This is returned as 7 characters in the form CYYMMDD, where:

C Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
YY Year
MM Month
DD Day

Default packet size (DFTPKTSIZE). The default packet size to use on the virtual circuit represented by this controller. This information is returned in two separate fields:

- *Transmit*
- *Receive*

(See the DFTPKTSIZE keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -10 implies *TRANSMIT
- Value of -16 implies *LIND

Default window size (DFTWDWSIZE). The default window size to use on the virtual circuit represented by this controller. This information is returned in two separate fields:

- *Transmit*
- *Receive*

(See the DFTWDWSIZE keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -10 implies *TRANSMIT
- Value of -16 implies *LIND

Destination service access point (DSAP). The logical address that this system will send to when it communicates with the remote controller. (See the DSAP keyword in the *OS/400* Communications Configuration Reference*.)

Device category. This value will be one of the following:

*APPC
*ASC
*BSC
*DKT
*DSP
*FNC
*HOST
*INTR
*NET
*PRT
*RTL
*SNPT
*SNUF
*TAP

The category value is derived from the command used to create the device description.

Device name. The name of a device associated with this controller.

Device text description. A brief description of a device associated with this controller.

Device type (TYPE). The type of device being described. For a full list of the possible type values, see the TYPE keyword in the *OS/400* Communications Configuration Reference*.)

Device wait timer (DEVWAITTMR). The device wait time-out value. (See the DEVWAITTMR keyword in the *OS/400* Communications Configuration Reference*.)

Dial initiation (DIALINIT). Whether or not the system should dial the remote system or controller immediately when this controller description is varied on. (See the DIALINIT keyword in the *OS/400* Communications Configuration Reference*.)

Dial retries (DIALRTY). The number of times to retry dialing the number before considering the dialing unsuccessful. (See the DIALRTY keyword in the *OS/400* Communications Configuration Reference*.)

Disconnect timer (DSCTMR). The options for automatic disconnection. These options are returned in two separate fields:

- *Minimum connect*: The minimum length of time the link stays active after the connection has been made.
- *Disconnect delay*: The length of time the system delays disconnection after the last session for the controller is ended.

(See the DSCTMR keyword in the *OS/400* Communications Configuration Reference*.)

Entry length for list of attached devices. Entry length in bytes of each element in the list of attached devices returned with this format. A value of zero is returned if the list is empty.

Entry length for list of remote identifiers. Entry length in bytes of each element in the list of remote identifiers returned with this format. A value of zero is returned if the list is empty.

Entry length for list of switched lines. Entry length in bytes of each element in the list of switched lines returned with this format. A value of zero is returned if the list is empty.

Exchange identifier (EXCHID). A hexadecimal value used to identify the remote controller. (See the EXCHID keyword in the *OS/400* Communications Configuration Reference*.)

File transfer acknowledgement timer (ACKTMR). The time allowed, in seconds, for an acknowledgement when using file transfer support. (See the ACKTMR keyword in the *OS/400* Communications Configuration Reference*.)

File transfer retry (RETRY). The number of attempts to transmit a frame after an unsuccessful transmission when using file transfer support. (See the RETRY keyword in the *OS/400* Communications Configuration Reference*.)

IDLC connect retry (IDLCCNNRTY). The number of times to retry a transmission at connection time. (See the IDLCCNNRTY keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -8 implies *NOMAX
- Value of -16 implies *LIND

IDLC default window size (IDLCWDWSIZ). The default window size used for this line description. (See the IDLCWDWSIZ keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -16 implies *LIND

IDLC frame retry (IDLCFRMRTY). The maximum number of transmissions to attempt before reporting an error. (See the IDLCFRMRTY keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -16 implies *LIND

IDLC response timer (IDLCRSPTMR). The length of time to wait before retransmitting a frame when an acknowledgement is not received. (See the IDLCRSPTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -16 implies *LIND

Initial connection (INLCNN). Whether the initial switched connection is made by the system when it answers an incoming call or by a call started from the system. (See the INLCNN keyword in the *OS/400* Communications Configuration Reference*.)

LAN access priority (LANACCPTY). The priority set in the actual frames that the system will send to the remote controller. (See the LANACCPTY keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

LAN acknowledgement frequency (LANACKFRQ). The maximum number of frames the system can receive before sending an acknowledgement to the remote controller. (See the LANACKFRQ keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

LAN acknowledgement timer (LANACKTMR). The length of time the system will delay before sending an acknowledgement to the remote controller for a received data frame. (See the LANACKTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

LAN connection retry (LANCNNRTY). The number of times a frame will be retransmitted during the connection establishment if there is no acknowledgement from the remote controller in the time period specified by the LANCNNTMR parameter. (See the LANCNNRTY keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

LAN connection timer (LANCNNTMR). The length of time to wait for an acknowledgement from the remote controller during the connection establishment before retransmitting a frame. (See the LANCNNTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

LAN frame retry (LANFRMRTY). The number of times a frame will be retransmitted if there is no acknowledgement from the remote controller in the time period specified by the LANRSPTMR parameter. (See the LANFRMRTY keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

LAN inactivity timer (LANINACTMR). The length of time that the system will wait for a frame from the remote controller before requesting data with a frame of its own. (See the LANINACTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

LAN maximum outstanding frames (LANMAXOUT). The maximum number of outstanding frames that the system sends to the remote controller before it waits for an acknowledgment. (See the LANMAXOUT keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

LAN remote adapter address (ADPTADR). The adapter address of the remote controller. (See the ADPTADR keyword in the *OS/400* Communications Configuration Reference*.)

LAN response timer (LANRSPTMR). The length of time to wait for an acknowledgement from the remote controller before retransmitting a data frame. (See the LANRSPTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

LAN window step (LANWDWSTP). Whether the number of outstanding frames (frames sent without receiving an acknowledgement from the remote system) should be reduced during periods of network congestion. (See the LANWDWSTP keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -3 implies *NONE

Link type (LINKTYPE). The type of line this controller will be attached to. (See the LINKTYPE keyword in the *OS/400* Communications Configuration Reference*.)

Local exchange identifier (LCLEXCHID). A hexadecimal value used to identify the local system to the remote system. (See the LCLEXCHID keyword in the *OS/400* Communications Configuration Reference*.)

Local identifier (LCLID). The name that, when combined with the local location name, identifies your controller to a remote system. (See the LCLID keyword in the *OS/400* Communications Configuration Reference*.)

Local location name (LCLLOCNAME). The name which, when combined with the local identifier, identifies your controller to a remote system. (See the LCLLOCNAME keyword in the *OS/400* Communications Configuration Reference*.)

Maximum frame size (MAXFRAME). The maximum path information unit (PIU) size that the controller can send or receive. (See the MAXFRAME keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -17 implies *LINKTYPE

Model controller description (MDLCTL). Whether this controller description is to be used as a model controller for automatically created controller descriptions associated with the line description specified on the SWTLINLST parameter. (See the MDLCTL keyword in the *OS/400* Communications Configuration Reference*.)

Negotiated packet size. The default packet size to use on the virtual circuit represented by this controller. This information is returned in two separate fields:

- *Transmit*
- *Receive*

(See the DFTPKTSIZE keyword in the *OS/400* Communications Configuration Reference*.)

Negotiated window size. The default window size to use on the virtual circuit represented by this controller. This information is returned in two separate fields:

- *Transmit*
- *Receive*

(See the DFTWDWSIZE keyword in the *OS/400* Communications Configuration Reference*.)

Number of attached devices. The number of elements in the list of attached devices returned with this format. A value of zero is returned if the list is empty.

Number of remote identifiers. The number of elements in the list of remote identifiers returned with this format. A value of zero is returned if the list is empty.

Number of switched lines. The number of elements in the list of switched lines returned with this format. A value of zero is returned if the list is empty.

Offset to list of attached devices. The offset in bytes to the first element in the list of attached devices returned with this format. A value of zero is returned if the list is empty.

Offset to list of remote identifiers. The offset in bytes to the first element in the list of remote identifiers returned with this format. A value of zero is returned if the list is empty.

Offset to list of switched lines. The offset in bytes to the first element in the list of switched lines returned with this format. A value of zero is returned if the list is empty.

Online at IPL (ONLINE). Whether the controller is varied on automatically when the system is turned on or you want to vary it on manually by using the Vary Configuration (VRYCFG) command. (See the ONLINE keyword in the *OS/400* Communications Configuration Reference*.)

PAD emulation (PADEML). Whether or not this controller is to emulate an X.25 packet assembler/disassembler (PAD). (See the PADEML keyword in the *OS/400* Communications Configuration Reference*.)

Predial delay (PREDIALDLY). The length of time to wait before dialing the number to establish a connection to the specified controller. (See the PREDIALDLY keyword in the *OS/400* Communications Configuration Reference*.)

Recontact at vary off (RECONTACT). Whether a recontact request is to be sent to the host system when this controller is varied off normally. (See the RECONTACT keyword in the *OS/400* Communications Configuration Reference*.)

Recovery limits (CMNRCYLMT). The second-level communications recovery limit for each controller description. These limits are returned in two separate fields:

- *Count limit*: The number of second-level recovery attempts to be automatically performed by the system.
- *Time interval*: The length of time (in minutes) in which the specified number of second-level recoveries can be attempted.

(See the CMNRCYLMT keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -14 implies *SYSVAL

Redial delay (REDIALDLY). The length of time to wait before redialing the number to establish a connection to the specified controller if the previous attempt was unsuccessful. (See the REDIALDLY keyword in the *OS/400* Communications Configuration Reference*.)

Remote control point name (RMTCPNAME). The name of the remote control point. (See the RMTCPNAME keyword in the *OS/400* Communications Configuration Reference*.)

Remote identifier. The identifier used to identify the remote system to the local system.

Remote location name (RMTLOCNAME). The name by which the remote work station controller is known to the network. (See the RMTLOCNAME keyword in the *OS/400* Communications Configuration Reference*.)

Remote network identifier (RMTNETID). The name of the remote network in which the adjacent control point resides. (See the RMTNETID keyword in the *OS/400* Communications Configuration Reference*.)

Remote verify (RMTVFY). Whether a remote system requires verification if a generic controller and device are configured to accept calls from any X.25 network address.

(See the RMTVFY keyword in the *OS/400* Communications Configuration Reference*.)

Reserved. Space included for alignment.

Resource name (RSRCNAME). The unique name that is assigned by the system to the physical equipment (in this case, a communications port) attached to the system. (See the RSRCNAME keyword in the *OS/400* Communications Configuration Reference*.)

RJE host type (RJEHOST). The name of the host system used by remote job entry. (See the RJEHOST keyword in the *OS/400* Communications Configuration Reference*.)

RJE host signon/logon (RJELOGON). The logon information required by the host system if you specified remote job entry as the application type. (See the RJELOGON keyword in the *OS/400* Communications Configuration Reference*.)

SDLC connect poll retry (CNNPOLLRTY). The number of connect poll retries that will be attempted before the AS/400 system indicates an error in contacting the remote system. (See the CNNPOLLRTY keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -8 implies *NOMAX
- Value of -11 implies *CALC

SDLC NDM poll timer (NDMPOLLTMR). The minimum interval at which a secondary station should be polled if a poll from the primary to the secondary station (which is in normal disconnect mode) does not result in receiving the appropriate response. (See the NDMPOLLTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

SDLC out limit (OUTLMT). The number of additional frame sequences the AS/400 system will send to the controller before polling the next station in the poll list. (See the OUTLMT keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -18 implies *POLLLMT

SDLC poll limit (POLLLMT). The number of additional consecutive polls that the AS/400 system will send to a controller when that controller responds by sending a number of frames equal to the maximum outstanding frames (MAXOUT parameter) specified on the line description. (See the POLLLMT keyword in the *OS/400* Communications Configuration Reference*.)

SDLC poll priority (POLLPTY). Whether this controller should have priority when being polled. (See the POLLPTY keyword in the *OS/400* Communications Configuration Reference*.)

Short-hold mode (SHM). Whether this controller is used for X.21 short-hold mode. (See the SHM keyword in the *OS/400* Communications Configuration Reference*.)

Short-hold mode disconnect limit (SHMDSCLMT). The number of consecutive nonproductive responses (RR or RNR) that are required from the remote station before the connection can be suspended for this X.21 short-hold mode connection. (See the SHMDSCLMT keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -8 implies *NOMAX

Short-hold mode disconnect timer (SHMDSCTMR). The minimum length of time that the primary station will maintain the connection to the remote system for this X.21 short-hold mode controller. (See the SHMDSCTMR keyword in the *OS/400* Communications Configuration Reference*.)

Source service access point (SSAP). The logical address this system will use when it sends data to the remote controller. (See the SSAP keyword in the *OS/400* Communications Configuration Reference*.)

Station address (STNADR). The station address to be used when communicating with the remote system using SDLC. (See the STNADR keyword in the *OS/400* Communications Configuration Reference*.)

Switched connection. Whether this controller is attached to a switched line, a local area network, or an X.25 switched virtual circuit (SVC) connection. (See the SWITCHED keyword in the *OS/400* Communications Configuration Reference*.)

Switched disconnect (SWTDSC). Whether the switched connection is dropped when the last session is unbound and the disconnect timer (DSCTMR) has ended. (See the SWTDSC keyword in the *OS/400* Communications Configuration Reference*.)

Switched line. Whether this controller is attached to a switched line. (See the SWITCHED keyword in the *OS/400* Communications Configuration Reference*.)

Switched line name. The name of a line that can be connected to this controller for switched connections. (See the SWTLINLST keyword in the *OS/400* Communications Configuration Reference*.)

Switched network backup (SNBU). Whether you want the switched network backup capability. (See the SNBU keyword in the *OS/400* Communications Configuration Reference*.)

System service control point identifier (SSCPID). The system service control point identifier that the AS/400 system sends to the remote system in the activate physical unit (ACTPU) request. (See the SSCPID keyword in the *OS/400* Communications Configuration Reference*.)

TDLC line name. The name of a line associated with this controller.

Text description (TEXT). A brief description of the controller and its location. (See the TEXT keyword in the *OS/400* Communications Configuration Reference*.)

Time information retrieved. The time that the information was provided by the API. It is returned as 6 characters in the form HHMMSS, where:

HH Hour
MM Minute
SS Second

User facilities (USRFCL). Allows network subscribers to request network-supplied facilities that are not available through the AS/400 parameters. (See the USRFCL keyword in the *OS/400* Communications Configuration Reference*.)

User-defined 1, 2, and 3 (USRDFN1, USRDFN2, USRDFN3). Used to describe any unique characteristics of this connection that you want to control. (See the USRDFN1, USRDFN2, and USRDFN3 keywords in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -16 implies *LIND

X.25 acknowledgement timer (X25ACKTMR). The ELLC LT2 acknowledgement timer, which is only used for controllers that have the X.25 link protocol (LINKPCL) set to *ELLC. (See the X25ACKTMR keyword in the *OS/400* Communications Configuration Reference*.)

X.25 connection password. For X.25 SVC connections, the password used when connecting to this controller.

X.25 connection retry (X25CNNRTY). Same as the X25FRMRTY parameter, except that it applies only to logical link control (LLC) connection establishment, such as LSABME-LUA LLC protocol data units for ELLC and QSM-QUA for QLLC LLC protocol data units. (See the X25CNNRTY keyword in the *OS/400* Communications Configuration Reference*.)

X.25 connection timer (X25CNNTMR). Same as the X25RSPTMR parameter, except that it applies only to LLC connection establishment, such as LSABME-LUA LLC protocol data units for ELLC and QSM-QUA LLC protocol data units for QLLC. (See the X25CNNTMR keyword in the *OS/400* Communications Configuration Reference*.)

X.25 delayed connection timer (X25DLYTMR). The time between retries of polling exchange identifier commands when the system is trying to establish a connection to the remote data terminal equipment (DTE) represented by the controller description. (See the X25DLYTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -11 implies *CALC

X.25 frame retry (X25FRMRTY). The number of times that a data or logical link disconnection protocol data unit (PDU) can be retransmitted if no acknowledgement is received from the adjacent logical link station in the remote DTE in the time specified by the X.25 response timer (X25RSPTMR). (See the X25FRMRTY keyword in the *OS/400* Communications Configuration Reference*.)

X.25 inactivity timer (X25INACTMR). The ELLC LTI inactivity timer, which is only used for controllers that have the X.25 link protocol (LINKPCL) set to *ELLC. (See the X25INACTMR keyword in the *OS/400* Communications Configuration Reference*.)

X.25 link protocol (LINKPCL). The logical link protocol to be used to communicate with the remote DTE represented by this controller description. (See the LINKPCL keyword in the *OS/400* Communications Configuration Reference*.)

X.25 logical channel ID (LGLCHLID). The logical channel identifier that is to be used for this controller. (See the LGLCHLID keyword in the *OS/400* Communications Configuration Reference*.)

X.25 network level (NETLVL). The level of the support by the X.25 network and the remote DTE represented by this controller description. (See the NETLVL keyword in the *OS/400* Communications Configuration Reference*.)

X.25 response timer (X25RSPTMR). The maximum amount of time allowed between the transmission of a data or logical link disconnection link protocol data unit (PDU) and the receipt of a corresponding acknowledgement from the adjacent link station on the remote DTE. (See the X25RSPTMR keyword in the *OS/400* Communications Configuration Reference*.)

X.25 reverse charging (RVSCRG). For incoming calls, whether reverse charging will be accepted, and for outgoing calls, whether reverse charging will be requested. (See the RVSCRG keyword in the *OS/400* Communications Configuration Reference*.)

X.25 switched line selection (SWTLINSLCT). Which of the lines listed on the SWTLINLST parameter will be selected for making the switched connection. (See the SWTLINSLCT keyword in the *OS/400* Communications Configuration Reference*.)

X.25 user group ID (USRGRPID). A value that is supplied as a unique identifier by the network if the closed user group facility is subscribed to. (See the USRGRPID keyword in the *OS/400* Communications Configuration Reference*.)

Error Messages

CPF24B4 E Severe error addressing parameter list.
CPF2625 E Not able to allocate object &1.
CPF2634 E Not authorized to object &1.
CPF268B E Option is not valid for controller.
CPF26A7 E Category of object incompatible with API format.

| CPF2702 E Device description &1 not found.
| CPF2703 E Controller description &1 not found.
| CPF3C19 E Error occurred with receiver variable specified.
| CPF3C21 E Format name &1 not valid.
| CPF3C24 E Length of receiver variable not valid.
| CPF3CF1 E Error code parameter not valid.
| CPF8104 E Controller description &4 damaged.
| CPF8105 E Device description &4 damaged.
| CPF8125 E Line description &4 damaged.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Device Description (QDCRDEVD) 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	Device name	Input	Char(10)
5	Error code	I/O	Char(*)

The Retrieve Device Description (QDCRDEVD) API retrieves information about a device description.

Authorities and Locks

Device Description Authority *USE
Device Description Lock *EXCLRD

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the device information.

Length of receiver variable
INPUT; BINARY(4)
The length of the area referenced by the receiver variable parameter. If the amount of information to be returned is greater than this value, the information will be truncated to this length.

Format name
INPUT; CHAR(8)
The content and format of the information returned for each device description. The possible format names are:

DEVD0100 Basic device information.
DEVD0200 Detailed information for device category *APPC
DEVD0300 Detailed information for device category *ASC
DEVD0400 Detailed information for device category *BSC
DEVD0500 Detailed information for device category *DKT
DEVD0600 Detailed information for device category *DSP
DEVD0700 Detailed information for device category *FNC
DEVD0800 Detailed information for device category *HOST
DEVD0900 Detailed information for device category *INTR
DEVD1000 Detailed information for device category *NET
DEVD1100 Detailed information for device category *PRT
DEVD1200 Detailed information for device category *RTL
DEVD1300 Detailed information for device category *SNPT
DEVD1400 Detailed information for device category *SNUF
DEVD1500 Detailed information for device category *TAP

See "Format of Device Information" for a description of these formats.

Device name
INPUT; CHAR(10)
The name of the device description to be retrieved.

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 Device Information

When the device category is unknown, specify DEVD0100 and the basic information (including device category) will be returned. When the device category is known, specify one of the other category-specific formats.

For detailed descriptions of the fields returned in these formats, see "Field Descriptions" on page 9-28.

DEVD0100 Format: Use this format to find out the device category, plus some basic information about the device. Then you may use the returned device category to select one of the other (category-specific) formats to call the API again for detailed information about the device description.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(7)	Date information retrieved
15	F	CHAR(6)	Time information retrieved

Offset		Type	Field
Dec	Hex		
21	15	CHAR(10)	Device name
31	1F	CHAR(10)	Device category
41	29	CHAR(10)	Online at IPL
51	33	CHAR(50)	Text description
101	65	CHAR(3)	Reserved

DEVD0200 Format: This format returns detailed information about a device of category *APPC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Offset to list of mode names
108	6C	BINARY(4)	Number of mode names
112	70	BINARY(4)	Entry length for list of mode names
116	74	CHAR(10)	Remote location name
126	7E	CHAR(10)	Local location name
136	88	CHAR(10)	Remote network identifier
146	92	CHAR(10)	Attached nonswitched controller name
156	9C	CHAR(10)	Message queue: name
166	A6	CHAR(10)	Message queue: library
176	B0	CHAR(10)	Local location address
186	BA	CHAR(10)	APPN capable
196	C4	CHAR(10)	Single session: indication
206	CE	CHAR(10)	Single session: number of conversations
216	D8	CHAR(10)	Locally controlled session
226	E2	CHAR(10)	Pre-established session
236	EC	CHAR(10)	Secure location
246	F6	CHAR(10)	Automatically configured
These fields repeat for each mode name		CHAR(10)	Mode name
		CHAR(2)	Reserved

DEVD0300 Format: This format returns detailed information about a device of category *ASC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	CHAR(10)	Remote location name
114	72	CHAR(10)	Attached nonswitched controller name

DEVD0400 Format: This format returns detailed information about a device of category *BSC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Record length
108	6C	BINARY(4)	Block length
112	70	CHAR(10)	Local location address
122	7A	CHAR(10)	Remote location name
132	84	CHAR(10)	Attached nonswitched controller name
142	8E	CHAR(10)	Connection type
152	98	CHAR(10)	Application type
162	A2	CHAR(10)	Contention resolution winner
172	AC	CHAR(10)	Blocking type
182	B6	CHAR(10)	Separator character
192	C0	CHAR(10)	Remote BSCEL
202	CA	CHAR(10)	Transmit in transparent mode
212	D4	CHAR(10)	Compress and decompress data
222	DE	CHAR(10)	Truncate trailing blanks
232	E8	CHAR(10)	Group separator type
242	F2	CHAR(10)	Emulated device
252	FC	CHAR(10)	Emulated keyboard
262	106	CHAR(10)	Emulated numeric lock
272	110	CHAR(10)	Emulated work station

DEVD0500 Format: This format returns detailed information about a device of category *DKT.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	CHAR(10)	Device type
114	72	CHAR(10)	Device model
124	7C	CHAR(10)	Resource name

DEVD0600 Format: This format returns detailed information about a device of category *DSP.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Character identifier: graphic character set
108	6C	BINARY(4)	Character identifier: code page
112	70	BINARY(4)	Maximum length of request unit
116	74	BINARY(4)	Inactivity timer
120	78	BINARY(4)	DBCS feature: RAM size
124	7C	BINARY(4)	Activation timer
128	80	BINARY(4)	Switch setting
132	84	BINARY(4)	Device port
136	88	BINARY(4)	Maximum outstanding frames
140	8C	BINARY(4)	Idle timer
144	90	BINARY(4)	NRM poll timer
148	94	BINARY(4)	Frame retry
152	98	BINARY(4)	Offset to list of auxiliary devices
156	9C	BINARY(4)	Number of auxiliary devices
160	A0	BINARY(4)	Entry length for list of auxiliary devices
164	A4	CHAR(10)	Device class
174	AE	CHAR(10)	Device type
184	B8	CHAR(10)	Device model
194	C2	CHAR(10)	Local location address
204	CC	CHAR(10)	Attached nonswitched controller name
214	D6	CHAR(10)	Keyboard language type
224	E0	CHAR(10)	Drop line at signoff
234	EA	CHAR(10)	Allow blinking cursor
244	F4	CHAR(10)	Print device
254	FE	CHAR(10)	Remote location name
264	108	CHAR(10)	Local location name
274	112	CHAR(10)	Remote network identifier
284	11C	CHAR(10)	Control session device description
294	126	CHAR(10)	Associated printer: name
304	130	CHAR(10)	Associated printer: remote network identifier
314	13A	CHAR(10)	Alternate printer: name
324	144	CHAR(10)	Alternate printer: remote network identifier
334	14E	CHAR(10)	Output queue: name
344	158	CHAR(10)	Output queue: library
354	162	CHAR(10)	Printer
364	16C	CHAR(10)	Print file: name
374	176	CHAR(10)	Print file: library
384	180	CHAR(10)	Work station customizing object: name
394	18A	CHAR(10)	Work station customizing object: library
404	194	CHAR(10)	Application type
414	19E	CHAR(10)	DBCS feature: matrix size
424	1A8	CHAR(10)	DBCS feature: language ID
434	1B2	CHAR(10)	DBCS feature: last code point
444	1BC	CHAR(10)	SNA pass-through device
454	1C6	CHAR(10)	SNA pass-through group name
464	1D0	CHAR(10)	Emulated device
474	1DA	CHAR(10)	Emulated device model
484	1E4	CHAR(10)	Emulating ASCII device
494	1EE	CHAR(10)	Physical attachment
504	1F8	CHAR(10)	Line speed
514	202	CHAR(10)	Word length
524	20C	CHAR(10)	Parity type
534	216	CHAR(10)	Stop bits
544	220	CHAR(20)	ASCII terminal identifier
564	234	CHAR(10)	Associated APPC device
574	23E	CHAR(256)	Host signon/logon command
830	33E	CHAR(2)	Reserved
These fields repeat for each auxiliary device		BINARY(4)	Auxiliary device address
		CHAR(10)	Auxiliary device type
		CHAR(2)	Reserved

DEVD0700 Format: This format returns detailed information about a device of category *FNC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Maximum length of request unit
108	6C	BINARY(4)	Inactivity timer
112	70	BINARY(4)	Activation timer
116	74	CHAR(10)	Device type
126	7E	CHAR(10)	Local location address
136	88	CHAR(10)	Remote location name

Offset		Type	Field
Dec	Hex		
146	92	CHAR(10)	Attached nonswitched controller name
156	9C	CHAR(10)	Device class
166	A6	CHAR(10)	SNA pass-through device
176	B0	CHAR(10)	SNA pass-through group name

DEVD0800 Format: This format returns detailed information about a device of category *HOST.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Maximum length of request unit
108	6C	CHAR(10)	Local location address
118	76	CHAR(10)	Remote location name
128	80	CHAR(10)	Attached nonswitched controller name
138	8A	CHAR(10)	Application type
148	94	CHAR(10)	Emulated device
158	9E	CHAR(10)	Emulated device model
168	A8	CHAR(10)	Emulated keyboard
178	B2	CHAR(10)	Emulated numeric lock
188	BC	CHAR(10)	Emulated work station
198	C6	CHAR(10)	End session with host

DEVD0900 Format: This format returns detailed information about a device of category *INTR.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	CHAR(10)	Remote location name

DEVD1000 Format: This format returns detailed information about a device of category *NET.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	CHAR(10)	Network type
114	72	CHAR(10)	Attached nonswitched controller name

DEVD1100 Format: This format returns detailed information about a device of category *PRT.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Font identifier: point size
108	6C	BINARY(4)	Maximum length of request unit
112	70	BINARY(4)	Pacing
116	74	BINARY(4)	Maximum pending requests
120	78	BINARY(4)	Print request timer
124	7C	BINARY(4)	Character identifier: graphic character set
128	80	BINARY(4)	Character identifier: code page
132	84	BINARY(4)	DBCS feature: RAM size
136	88	BINARY(4)	Inactivity timer
140	8C	BINARY(4)	Activation timer
144	90	BINARY(4)	Switch setting
148	94	BINARY(4)	Device port
152	98	CHAR(10)	Device class
162	A2	CHAR(10)	Device type
172	AC	CHAR(10)	Device model
182	B6	CHAR(10)	Advanced Function Printing
192	C0	CHAR(10)	AFP attachment
202	CA	CHAR(10)	Local location address
212	D4	CHAR(10)	Attached nonswitched controller name
222	DE	CHAR(10)	Font identifier: identifier
232	E8	CHAR(10)	Form feed
242	F2	CHAR(10)	Separator drawer
252	FC	CHAR(10)	Printer error message
262	106	CHAR(10)	Message queue: name
272	110	CHAR(10)	Message queue: library
282	11A	CHAR(10)	Application type
292	124	CHAR(10)	Print while converting
302	12E	CHAR(10)	Form definition: name
312	138	CHAR(10)	Form definition: library
322	142	CHAR(10)	Work station customizing object: name
332	14C	CHAR(10)	Work station customizing object: library
342	156	CHAR(10)	Remote location name
352	160	CHAR(10)	Local location name
362	16A	CHAR(10)	Remote network identifier

Offset		Type	Field
Dec	Hex		
372	174	CHAR(10)	Control session device description
382	17E	CHAR(10)	Mode name
392	188	CHAR(10)	DBCS feature: matrix size
402	192	CHAR(10)	DBCS feature: language ID
412	19C	CHAR(10)	DBCS feature: last code point
422	1A6	CHAR(10)	SNA pass-through device
432	1B0	CHAR(10)	SNA pass-through group name
442	1BA	CHAR(10)	Emulated device
452	1C4	CHAR(10)	Emulated device model
462	1CE	CHAR(10)	Emulating ASCII device
472	1D8	CHAR(10)	Physical attachment
482	1E2	CHAR(10)	Auxiliary printer
492	1EC	CHAR(10)	Language type
502	1F6	CHAR(10)	Line speed
512	200	CHAR(10)	Word length
522	20A	CHAR(10)	Parity type
532	214	CHAR(10)	Stop bits
542	21E	CHAR(10)	Number of drawers
552	228	CHAR(10)	Print quality
562	232	CHAR(10)	Transform enabled
572	23C	CHAR(20)	Manufacturer type and model
592	250	CHAR(10)	Paper source 1
602	25A	CHAR(10)	Paper source 2
612	264	CHAR(10)	Envelope source
622	26E	CHAR(10)	ASCII code page 899 support
632	278	CHAR(10)	Separator exit program: name
642	282	CHAR(10)	Separator exit program: library
652	28C	CHAR(256)	Host signon/logon command

DEVD1200 Format: This format returns detailed information about a device of category *RTL.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Pacing
108	6C	BINARY(4)	Maximum length of request unit
112	70	BINARY(4)	Inactivity timer
116	74	BINARY(4)	Activation timer
120	78	CHAR(10)	Local location address
130	82	CHAR(10)	Remote location name
140	8C	CHAR(10)	Attached nonswitched controller name
150	96	CHAR(10)	Application type
160	A0	CHAR(10)	Device class
170	AA	CHAR(10)	SNA pass-through device
180	B4	CHAR(10)	SNA pass-through group name

DEVD1300 Format: This format returns detailed information about a device of category *SNPT.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Activation timer
108	6C	CHAR(10)	Local location address
118	76	CHAR(10)	SNA pass-through class
128	80	CHAR(10)	Attached nonswitched controller name
138	8A	CHAR(10)	SNA pass-through device
148	94	CHAR(10)	SNA pass-through group name

DEVD1400 Format: This format returns detailed information about a device of category *SNUF.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Record length
108	6C	BINARY(4)	Block length
112	70	CHAR(10)	Local location address
122	7A	CHAR(10)	Remote location name
132	84	CHAR(10)	Attached nonswitched controller name
142	8E	CHAR(10)	Program start request capable
152	98	CHAR(10)	Special host application
162	A2	CHAR(10)	Application identifier
172	AC	CHAR(10)	Host type
182	B6	CHAR(10)	Default program: name

Offset			
Dec	Hex	Type	Field
192	C0	CHAR(10)	Default program: library
202	CA	CHAR(10)	HCP emulation

DEVD1500 Format: This format returns detailed information about a device of category *TAP.

Offset			
Dec	Hex	Type	Field
0	0		Returns everything from format DEVD0100
104	68	BINARY(4)	Switch setting
108	6C	CHAR(10)	Device type
118	76	CHAR(10)	Device model
128	80	CHAR(10)	Resource name
138	8A	CHAR(10)	Message queue: name
148	94	CHAR(10)	Message queue: library
158	9E	CHAR(10)	Attached nonswitched controller name
168	A8	CHAR(10)	Assign device at vary on
178	B2	CHAR(10)	Unload device at vary off

Field Descriptions

Fields in the various formats returned by this API are described in the chapter on communications device descriptions in the *OS/400* Communications Configuration Reference* and in the *Device Configuration Guide*.

In certain cases, numeric values are assigned by this API to represent character values for some of the returned fields. Where a numeric value is assigned, the numeric value and the equivalent character value are listed as an *Exception* in the following field descriptions.

Activation timer (ACTTMR). The number of seconds the system should wait for the device to respond to an activation request from the host. (See the ACTTMR keyword in the *OS/400* Communications Configuration Reference*.)

Advanced Function Printing (AFP). Whether this printer is used for Advanced Function Printing support. (See the AFP keyword in the *OS/400* Communications Configuration Reference*.)

AFP attachment (AFPATTACH). The type of attachment used for printers configured for Advanced Function Printing support. (See the AFPATTACH keyword in the *OS/400* Communications Configuration Reference*.)

Allow blinking cursor (ALWBLN). Whether or not the cursor will blink for display devices. (See the ALWBLN keyword in the *OS/400* Communications Configuration Reference*.)

Alternate printer: name. The name of the secondary printer specified for a session.

Alternate printer: remote network identifier. The name of the remote network identifier for the alternate printer. (See the RMTNETID keyword in the *OS/400* Communications Configuration Reference*.)

Application identifier (APPID). The VTAM application identifier of the CICS/VS or IMS/VS host subsystem with which the AS/400 system communicates. (See the APPID keyword in the *OS/400* Communications Configuration Reference*.)

Application type (APPTYPE). The application type used by this device. (See the APPTYPE keyword in the *OS/400* Communications Configuration Reference*.)

APPN capable (APPN). Whether or not networking is used. (See the APPN keyword in the *OS/400* Communications Configuration Reference*.)

ASCII code page 899 support. Whether this printer has ASCII code page 899 installed.

ASCII terminal identifier. A user-specified terminal identifier for the physical device.

Assign device at vary on. Whether the tape device is assigned to the system when it is varied on. (See the ASSIGN keyword in the *OS/400* Communications Configuration Reference*.)

Associated APPC device. The name of an APPC device associated with a display station.

Associated printer: name. The name of the primary printer specified for a session.

Associated printer: remote network identifier. The name of the remote network identifier for the associated printer. (See the RMTNETID keyword in the *OS/400* Communications Configuration Reference*.)

Attached nonswitched controller name (CTL). The name of the controller description to which this device is attached. (See the CTL keyword in the *OS/400* Communications Configuration Reference*.)

Automatically configured. Whether this device has been configured automatically.

Auxiliary device address. The address of an additional device attached to an IEEE-48 port on this device. (See the AUXDEV keyword in the *OS/400* Communications Configuration Reference*.)

Auxiliary device type. The type of additional device attached to an IEEE-48 port on this device. (See the AUXDEV keyword in the *OS/400* Communications Configuration Reference*.)

Auxiliary printer. For ASCII printers, whether the printer is attached to a display station.

Block length (BLKLEN). The maximum block length (in bytes) for data to be transmitted when communicating with this device. (See the BLKLEN keyword in the *OS/400* Communications Configuration Reference*.)

Blocking type (BLOCK). Whether you or the AS/400 system will block and deblock transmitted records. (See the BLOCK keyword in the *OS/400* Communications Configuration Reference*.)

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 identifier (CHRID). The character identifier that this display station supports. This identifier has two parts, which are returned in separate fields:

- *Graphic character set*
- *Code page*

(See the CHRID keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -14 implies *SYSVAL
- Value of -27 implies *KBDTYPE

Compress and decompress data (DTACPR). Whether or not to have blanks in BSC data compressed for output and decompressed for input. (See the DTACPR keyword in the *OS/400* Communications Configuration Reference*.)

Connection type (CNN). The connection type. (See the CNN keyword in the *OS/400* Communications Configuration Reference*.)

Contention resolution winner (CTNWIN). Which BSC system is to be the primary unit and which is to be the secondary unit for contention resolution on a BSC line. (See the CTNWIN keyword in the *OS/400* Communications Configuration Reference*.)

Control session device description. Which control session device description created or selected a specific device description that is being used in a session.

Date information retrieved. The date that the information was provided by the API. This is returned as 7 characters in the form CYYMMDD, where:

C Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
YY Year
MM Month
DD Day

DBCS feature (IGCFEAT). The values that should be specified for DBCS display stations and printers. This information is returned in four separate fields:

- *RAM size*: The relative buffer size.
- *Matrix size*: The number of matrix points used to create the character.
- *Language ID*: A letter code that identifies the language used.
- *Last code point*: The code point of the last double-byte character.

(See the IGCFEAT keyword in the *OS/400* Communications Configuration Reference*.)

Default program (DFTPGM). The program to be called if a program start request is received from a host system that is not using an `*EXEC/*EXEX/*TXTC/*TXTX` format. This information is returned in two separate fields:

- *Name* of the program
- *Library* in which the program can be found

(See the DFTPGM keyword in the *OS/400* Communications Configuration Reference*.)

Device category. This value will be one of the following:

*APPC
*ASC
*BSC
*DKT
*DSP
*FNC
*HOST
*INTR
*NET
*PRT
*RTL
*SNPT
*SNUF
*TAP

The category value is derived from the command used to create the device description.

Device class (DEVCLS). The device class. (See the DEVCLS keyword in the *OS/400* Communications Configuration Reference*.)

Device model (MODEL). The model number of the device. (See the MODEL keyword in the *OS/400* Communications Configuration Reference*.)

Device name. The name of the device description.

Device port. The identification of the port to which this device is currently attached.

Device type (TYPE). The display device type. (See the TYPE keyword in the *OS/400* Communications Configuration Reference*.)

Drop line at signoff (DROP). Whether display stations attached to controllers on switched lines will be disconnected by the system when all work stations on the line are no longer being used. (See the DROP keyword in the *OS/400* Communications Configuration Reference*.)

Emulated device (EMLDEV). The type of device to be emulated. For devices of category *DSP and *PRT, this is a twinaxial device. For devices of category *HOST, this is a 3270 device. (See the EMLDEV keyword in the *OS/400* Communications Configuration Reference*.)

Emulated device model. The model of the device type to be emulated.

Emulated keyboard (EMLKBD). The type of 3278 display keyboard to be emulated. (See the EMLKBD keyword in the *OS/400* Communications Configuration Reference*.)

Emulated numeric lock (EMLNUMLCK). Whether numeric input fields allow only numeric data on a 5250 keyboard. (See the EMLNUMLCK keyword in the *OS/400* Communications Configuration Reference*.)

Emulated work station (EMLWRKSTN). The name of an emulated device associated with a real display station or printer device. (See the EMLWRKSTN keyword in the *OS/400* Communications Configuration Reference*.)

Emulating ASCII device. Whether this device is emulating a supported ASCII device type. (See the EMLASCII keyword in the *OS/400* Communications Configuration Reference*.)

End session with host. Whether a request-shutdown or unbind will be used to end a session.

Entry length for list of auxiliary devices. The entry length in bytes of each element in the list of auxiliary devices returned with this format. A value of zero is returned if the list is empty.

Entry length for list of mode names. The entry length in bytes of each element in the list of mode names returned with this format. A value of zero is returned if the list is empty.

Envelope source. The type of envelope to be used in paper source three.

Font identifier (FONT). The font identifier and point size used by *IPDS, 3812, and 5219 printers. This information is returned in two separate fields:

- *Font identifier*
- *Point size*

(See the FONT keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -3 implies *NONE

Form definition (FORMDF). The form definition to be used for print requests that do not specify a form definition. This information is returned in two separate fields:

- *Name* of the form definition
- *Library* in which the form definition can be found

(See the FORMDF keyword in the *OS/400* Communications Configuration Reference*.)

Form feed (FORMFEED). The mode in which forms are fed into the *IPDS, 4212, or 5219 printers. (See the FORMFEED keyword in the *OS/400* Communications Configuration Reference*.)

Frame retry (FRAMERTY). The number of retries for an unanswered command frame or an unacknowledged information frame. (See the FRAMERTY keyword in the *OS/400* Communications Configuration Reference*.)

Group separator type (GRPSEP). The separator for groups of data (data sets, documents, and so forth). (See the GRSEP keyword in the *OS/400* Communications Configuration Reference*.)

HCP emulation (HCLEML). The type of host command processor emulated session that this device description will be used for. (See the HCPEML keyword in the *OS/400* Communications Configuration Reference*.)

Host signon/logon command (LOGON). The logon string that is sent to the host network when the file is opened. (See the LOGON keyword in the *OS/400* Communications Configuration Reference*.)

Host type (HOST). The type of host system with which the device will communicate. (See the HOST keyword in the *OS/400* Communications Configuration Reference*.)

Idle timer (IDLTMR). The time that the system waits for a response. (See the IDLTMR keyword in the *OS/400* Communications Configuration Reference*.)

Inactivity timer (INACTTMR). The amount of time the device can be inactive before the session is ended. (See the INACTTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -8 implies *NOMAX
- Value of -19 implies *ATTACH
- Value of -20 implies *SEC15
- Value of -21 implies *SEC30

Keyboard language type (KBDTYPE). The 3-character keyboard type identified for type 3277, 3278, or 3279 display stations. (See the KBDTYPE keyword in the *OS/400* Communications Configuration Reference*.)

Language type. The keyboard language type for an ASCII printer.

| **Line speed (LINESPEED)**. A line speed for use with this device. (See the LINESPEED keyword in the *OS/400* Communications Configuration Reference*.)

| **Local location address (LOCADR)**. The local location address. (See the LOCADR keyword in the *OS/400* Communications Configuration Reference*.)

| **Local location name (LCLLOCNAME)**. The name by which the local AS/400 system is known to other devices in the network. (See the LCLLOCNAME keyword in the *OS/400* Communications Configuration Reference*.)

| **Locally controlled session (LCLCTLSSN)**. Whether the single session is locally or remotely controlled. (See the LCLCTLSSN keyword in the *OS/400* Communications Configuration Reference*.)

| **Manufacturer type and model**. The manufacturer, type, and model for a printer using transform support.

| **Maximum length of request unit (MAXLENRU)**. The default maximum size of the request/response unit (RU) that can be sent or received by the local system if the maximum size is not specified in the bind command received from the host system. (See the MAXLENRU keyword in the *OS/400* Communications Configuration Reference*.)

| *Exception:*

| - Value of -11 implies *CALC

| **Maximum outstanding frames (MAXOUT)**. The maximum number of frames that are sent to a remote system before it must respond. (See the MAXOUT keyword in the *OS/400* Communications Configuration Reference*.)

| **Maximum pending requests (MAXPNDRQS)**. The maximum number of print requests that can be queued for printers configured for Advanced Function Printing support. (See the MAXPNDRQS keyword in the *OS/400* Communications Configuration Reference*.)

| **Message queue (MSGQ)**. The message queue to which operational messages for this device are to be sent. This information is returned in two separate fields:

| - *Name* of the queue
| - *Library* in which the queue can be found

| (See the MSGQ keyword in the *OS/400* Communications Configuration Reference*.)

| **Mode name (MODE)**. The names used by the local AS/400 system and the remote system to refer to the group of sessions between the local and remote locations with the same characteristics. (See the MODE keyword in the *OS/400* Communications Configuration Reference*.)

| **Network type**. The type of network the device represents.

| **NRM poll timer (NRMPOLLTMR)**. The time interval for polling this device in normal response mode. (See the NRMPOLLTMR keyword in the *OS/400* Communications Configuration Reference*.)

| **Number of auxiliary devices**. The number of elements in the list of auxiliary devices returned with this format. A value of zero is returned if the list is empty.

| **Number of drawers**. The number of drawers the printer physically supports, not which drawer the paper is selected from. The individual print files sent to the printer determine which drawer is selected.

| **Number of mode names**. The number of elements in the list of mode names returned with this format. A value of zero is returned if the list is empty.

| **Offset to list of auxiliary devices**. The offset in bytes to the first element in the list of auxiliary devices returned with this format. A value of zero is returned if the list is empty.

| **Offset to list of mode names**. The offset in bytes to the first element in the list of mode names returned with this format. A value of zero is returned if the list is empty.

| **Online at IPL (ONLINE)**. The name that will be used when you are working with the Vary Configuration (VRYCFG) and Work with Configuration Status (WRKCFGSTS) commands. (See the ONLINE keyword in the *OS/400* Communications Configuration Reference*.)

| **Output queue (OUTQ)**. The output queue to be used for printed output associated with this display station. This information is returned in two separate fields:

| - *Name* of the queue
| - *Library* in which the queue can be found

| (See the OUTQ keyword in the *OS/400* Communications Configuration Reference*.)

| **Pacing (PACING)**. The number of request units (RUs) that can be sent or received before a pacing response must be sent or received. (See the PACING keyword in the *OS/400* Communications Configuration Reference*.)

| **Paper source 1**. The type of paper to be used in paper source one.

| **Paper source 2**. The type of paper to be used in paper source two.

| **Parity type (PARITY)**. For ASCII devices, the parity used to communicate over the attachment between the controller and the device. (See the PARITY keyword in the *OS/400* Communications Configuration Reference*.)

| **Physical attachment**. The attachment of a display station to the ASCII workstation controller. (See the ATTACH keyword in the *OS/400* Communications Configuration Reference*.)

| **Pre-established session (PREESTSSN)**. Whether the single session is to be established when connection with the

| remote system is established. (See the PREESTSSN
| keyword in the *OS/400* Communications Configuration
| Reference*.)

| **Print device (PRTDEV)**. The name of the printer to be used
| for printed output from this display device. (See the
| PRTDEV keyword in the *OS/400* Communications Config-
| uration Reference*.)

| **Print file (PRTFILE)**. The alternative printer device file to be
| used when no associated work station printer exists or when
| an error occurs during an attempt to use the work station
| printer. This information is returned in two separate fields:

- | *Name* of the device file
- | *Library* in which the device file can be found

| (See the PRTFILE keyword in the *OS/400* Communications
| Configuration Reference*.)

| **Print quality**. For printers, the quality of print to be
| produced.

| **Print request timer (PRTRQSTMR)**. The number of
| seconds to wait after a print request has been sent to a con-
| tinuous forms printer before the last printed output is forced
| into the output hopper. (See the PRTRQSTMR keyword in
| the *OS/400* Communications Configuration Reference*.)

| *Exception:*

- | Value of -8 implies *NOMAX

| **Print while converting (PRTCVT)**. Allows printers config-
| ured as AFP(*YES) to begin printing a spooled file while that
| file is being converted to an Advanced Function Printing data
| stream (AFPDS). (See the PRTCVT keyword in the *OS/400*
| Communications Configuration Reference*.)

| **Printer (PRINTER)**. The device name of the printer to be
| associated with the display device. (See the PRINTER
| keyword in the *OS/400* Communications Configuration
| Reference*.)

| **Printer error message (PRTERRMSG)**. Whether to have
| the printer send inquiry messages or informational messages
| for recoverable errors. (See the PRTERRMSG keyword in
| the *OS/400* Communications Configuration Reference*.)

| **Program start request capable (PGMSTRRQS)**. Whether
| or not to have the device reserved for program start
| requests. (See the PGMSTRRQS keyword in the *OS/400*
| Communications Configuration Reference*.)

| **Record length (RCDLEN)**. The maximum record length
| allowed when communicating with this device. (See the
| RCDLEN keyword in the *OS/400* Communications Config-
| uration Reference*.)

| **Remote BSCEL (RMTBSCEL)**. Whether this device will
| communicate with a remote system that can recognize
| BSCEL commands and messages. (See the RMTBSCEL
| keyword in the *OS/400* Communications Configuration
| Reference*.)

| **Remote location name (RMTLOCNAME)**. The name of the
| remote location with which your system will be communi-
| cating. (See the RMTLOCNAME keyword in the *OS/400*
| Communications Configuration Reference*.)

| **Remote network identifier (RMTNETID)**. The 8-character
| name of the remote network in which the location resides.
| (See the RMTNETID keyword in the *OS/400* Communica-
| tions Configuration Reference*.)

| **Reserved**. Space included for alignment.

| **Resource name**. The unique name that is assigned by the
| system to the physical equipment attached to the system.
| For an explanation of resource names for devices, see the
| *Device Configuration Guide*.

| **Secure location (SECURELOC)**. Whether or not the local
| location allows the remote location to verify user passwords
| and to send a verified indicator with the program start
| request. (See the SECURELOC keyword in the *OS/400*
| Communications Configuration Reference*.)

| **Separator character (SEPCHAR)**. The separator character
| to be used (if you specified *SEP for the blocking type).
| (See the SEPCHAR keyword in the *OS/400* Communica-
| tions Configuration Reference*.)

| **Separator drawer (SEPDRAWER)**. The sheet feeding
| drawer for file and job separators. (See the SEPDRAWER
| keyword in the *OS/400* Communications Configuration
| Reference*.)

| **Separator exit program (SEPPGM)**. A user exit program to
| be called when printing job and file separators. This informa-
| tion is returned in two separate fields:

- | *Name* of the program
- | *Library* in which the program can be found

| (See the SEPPGM keyword in the *OS/400* Communications
| Configuration Reference*.)

| **Single session (SNGSSN)**. Whether communications with
| the remote location is limited to one session. If communica-
| tions is limited to one session, a maximum number of con-
| versations may also be specified. This information is
| returned in two separate fields:

- | *Indication* of whether communications is limited to one
| session
- | *Number of conversations* allowed if limited to one
| session

| (See the SNGSSN keyword in the *OS/400* Communications
| Configuration Reference*.)

| **SNA pass-through class (SNPTCLS)**. Whether this device
| is to be used as an upstream or downstream pass-through
| device. (See the SNPTCLS keyword in the *OS/400* Com-
| munications Configuration Reference*.)

SNA pass-through device (SNAPTDEV). The name of the pass-through device with which this device is associated. (See the SNPTDEV keyword in the *OS/400* Communications Configuration Reference*.)

SNA pass-through group name (SNPTGAP). The name of a group of upstream SNA pass-through devices with which this device can be associated. (See the SNPTGRP keyword in the *OS/400* Communications Configuration Reference*.)

Special host application (SPCHOSTAPP). Whether this device is used to communicate with a special host application. (See the SPCHOSTAPP keyword in the *OS/400* Communications Configuration Reference*.)

Stop bits (STOPBITS). For ASCII devices, the number of stop bits used to communicate over the attachment between the controller and the device. (See the STOPBITS keyword in the *OS/400* Communications Configuration Reference*.)

Switch setting. The value of the current switch settings at the actual device, equivalent to the current device address.

Text description (TEXT). A brief description of the device and its location. (See the TEXT keyword in the *OS/400* Communications Configuration Reference*.)

Time information retrieved. The time that the information was provided by the API. It is returned as 6 characters in the form HHMMSS, where:

HH Hour
MM Minute
SS Second

Transform enabled. Whether this printer will use the host-based transform support to convert SCS to ASCII.

Transmit in transparent mode (TRNSPY). Whether transparency is to be used by this device. (See the TRNSPY keyword in the *OS/400* Communications Configuration Reference*.)

Truncate trailing blanks (TRUNC). Whether trailing blanks are to be removed from the output records. (See the TRUNC keyword in the *OS/400* Communications Configuration Reference*.)

Unload device at vary off. Whether the tape device will be unloaded when the device is varied off.

Word length. For ASCII devices, the word length used to communicate over the attachment between the controller and the device. (See the WORDLEN keyword in the *OS/400* Communications Configuration Reference*.)

Work station customizing object (WSCST). The object containing pointers to the work station customizing tables for this device. This information is returned in two separate fields:

- *Name* of the customizing object
- *Library* in which the object can be found

(See the WSCST keyword in the *OS/400* Communications Configuration Reference*.)

Error Messages

CPF24B4 E Severe error addressing parameter list.
CPF2625 E Not able to allocate object &1.
CPF2634 E Not authorized to object &1.
CPF26A7 E Category of object incompatible with API format.
CPF2702 E Device description &1 not found.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 not valid.
CPF3C24 E Length of receiver variable not valid.
CPF3CF1 E Error code parameter not valid.
CPF8104 E Controller description &4 damaged.
CPF8105 E Device description &4 damaged.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Line Description (QDCRLIND) 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	Line name	Input	Char(10)
5	Error code	I/O	Char(*)

The Retrieve Line Description (QDCRLIND) API retrieves information about a line description.

Authorities and Locks

Controller Description Authority *USE
Line Description Authority *USE
Controller Description Lock *EXCLRD
Line Description Lock *EXCLRD

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the line information.

Length of receiver variable
INPUT; BINARY(4)
The length of the area referenced by the receiver variable parameter. If the amount of information to be returned is greater than this value, the information will be truncated to this length.

Format name
INPUT; CHAR(8)
The content and format of the information returned for each line description. The possible format names are:

LIND0100 Basic line information
LIND0200 Basic line information, plus list of attached nonswitched controllers
LIND0300 Detailed information for line category *ASC
LIND0400 Detailed information for line category *BSC
LIND0500 Detailed information for line category *ETH
LIND0600 Detailed information for line category *IDLC
LIND0700 Detailed information for line category *NET
LIND0800 Detailed information for line category *SDLC
LIND0900 Detailed information for line category *TDLC
LIND1000 Detailed information for line category *TRN
LIND1100 Detailed information for line category *X25
LIND1200 Detailed information for line category *DDI
LIND1300 Detailed information for line category *FR
LIND1400 Detailed information for line category *FAX

See "Format of Line Information" for a description of these formats.

Line name
INPUT; CHAR(10)
The name of the line description to be retrieved.

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 Line Information

When the line category is unknown, specify LIND0100 or LIND0200, and the basic information (including line category) will be returned. When the line category is known, specify one of the other category-specific formats.

For detailed descriptions of the fields returned in these formats, see "Field Descriptions" on page 9-43.

LIND0100 Format: Use this format to find out the line category, plus some basic information about the line. Then you may use the returned line category to select one of the other (category-specific) formats to call the API again for detailed information about the line description. This format also returns the number of controllers currently attached to this line.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of attached nonswitched controllers
12	C	CHAR(7)	Date information retrieved
19	13	CHAR(6)	Time information retrieved
25	19	CHAR(10)	Line name
35	23	CHAR(10)	Line category
45	2D	CHAR(10)	Online at IPL
55	37	CHAR(50)	Text description
105	69	CHAR(3)	Reserved

LIND0200 Format: This format returns basic line information, plus a list of attached nonswitched controllers. Some basic information is also included for each attached nonswitched controller.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Offset to list of attached nonswitched controllers
112	70	BINARY(4)	Entry length for list of attached nonswitched controllers
These fields repeat for each nonswitched controller		CHAR(10)	Attached nonswitched controller name
		CHAR(10)	Controller category
		CHAR(10)	Controller type
		CHAR(50)	Controller text description

LIND0300 Format: This format returns detailed information about a line of category *ASC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
112	70	BINARY(4)	Line speed
116	74	BINARY(4)	Inactivity timer
120	78	BINARY(4)	Maximum buffer size
124	7C	BINARY(4)	Idle timer
128	80	BINARY(4)	Data Set Ready drop timer
132	84	BINARY(4)	Clear to Send timer
136	88	BINARY(4)	Remote answer timer
140	8C	BINARY(4)	Recovery limits: count limit
144	90	BINARY(4)	Recovery limits: time interval
148	94	BINARY(4)	Offset to list of attached nonswitched controllers

Offset		Type	Field
Dec	Hex		
152	98	BINARY(4)	Entry length for list of attached nonswitched controllers
156	9C	BINARY(4)	Offset to list of switched controllers
160	A0	BINARY(4)	Number of switched controllers
164	A4	BINARY(4)	Entry length for list of switched controllers
168	A8	BINARY(4)	Offset to list of active switched controllers
172	AC	BINARY(4)	Number of active switched controllers
176	B0	BINARY(4)	Entry length for list of active switched controllers
180	B4	BINARY(4)	Offset to list of EOR characters
184	B8	BINARY(4)	Number of EOR characters
188	BC	BINARY(4)	Entry length for list of EOR characters
192	C0	CHAR(10)	Resource name
202	CA	CHAR(10)	Physical Interface
212	D4	CHAR(10)	Connection type
222	DE	CHAR(10)	Switched network backup
232	E8	CHAR(10)	Activate switched network backup
242	F2	CHAR(10)	Autocall unit
252	FC	CHAR(10)	Data bits per character
262	106	CHAR(10)	Type of parity
272	110	CHAR(10)	Stop bits
282	11A	CHAR(10)	Duplex
292	124	CHAR(10)	Echo support
302	12E	CHAR(10)	Modem type supported
312	138	CHAR(10)	Modem data rate select
322	142	CHAR(10)	Switched connection type
332	14C	CHAR(10)	Autoanswer
342	156	CHAR(10)	Autodial
352	160	CHAR(10)	Dial command type
362	16A	CHAR(10)	Autocall resource name
372	174	CHAR(32)	Calling number
404	194	CHAR(10)	Error threshold level
414	19E	CHAR(10)	Flow control
424	1A8	CHAR(10)	XON character
434	1B2	CHAR(10)	XOFF character
444	1BC	CHAR(10)	Autoanswer type

Offset		Type	Field
Dec	Hex		
These fields repeat for each non-switched controller		CHAR(10)	Attached nonswitched controller name
		CHAR(2)	Reserved
These fields repeat for each switched controller		CHAR(10)	Switched controller name
		CHAR(2)	Reserved
These fields repeat for each active switched controller		CHAR(10)	Active switched controller name
		CHAR(2)	Reserved
These fields repeat for each EOR character		BINARY(4)	Number of trailing characters
		CHAR(10)	EOR character
		CHAR(2)	Reserved

LIND0400 Format: This format returns detailed information about a line of category *BSC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
112	70	BINARY(4)	Line speed
116	74	BINARY(4)	Inactivity timer
120	78	BINARY(4)	Maximum buffer size
124	7C	BINARY(4)	Receive timer
128	80	BINARY(4)	Continue timer
132	84	BINARY(4)	Contention state retry
136	88	BINARY(4)	Data state retry
140	8C	BINARY(4)	Transmit TTD or WACK retry
144	90	BINARY(4)	Receive TTD or WACK retry
148	94	BINARY(4)	Data Set Ready drop timer
152	98	BINARY(4)	Clear To Send timer
156	9C	BINARY(4)	Remote answer timer
160	A0	BINARY(4)	Recovery limits: count limit
164	A4	BINARY(4)	Recovery limits: time interval
168	A8	BINARY(4)	Offset to list of attached nonswitched controllers
172	AC	BINARY(4)	Entry length for list of attached nonswitched controllers

Offset		Type	Field
Dec	Hex		
176	B0	BINARY(4)	Offset to list of switched controllers
180	B4	BINARY(4)	Number of switched controllers
184	B8	BINARY(4)	Entry length for list of switched controllers
188	BC	BINARY(4)	Offset to list of active switched controllers
192	C0	BINARY(4)	Number of active switched controllers
196	C4	BINARY(4)	Entry length for list of active switched controllers
200	C8	CHAR(10)	Resource name
210	D2	CHAR(10)	Application type
220	DC	CHAR(10)	Physical Interface
230	E6	CHAR(10)	Connection type
240	F0	CHAR(10)	Switched network backup
250	FA	CHAR(10)	Activate switched network backup
260	104	CHAR(10)	Autocall unit
270	10E	CHAR(10)	Station address
280	118	CHAR(10)	Clocking
290	122	CHAR(10)	Duplex
300	12C	CHAR(10)	Modem type supported
310	136	CHAR(10)	Modem data rate select
320	140	CHAR(10)	Switched connection type
330	14A	CHAR(10)	Autoanswer
340	154	CHAR(10)	Autodial
350	15E	CHAR(10)	Dial command type
360	168	CHAR(10)	Autocall resource name
370	172	CHAR(32)	Calling number
402	192	CHAR(10)	Character code
412	19C	CHAR(10)	SYN characters
422	1A6	CHAR(10)	Error threshold level
432	1B0	CHAR(10)	Include STX character in the LRC
442	1BA	CHAR(10)	Autoanswer type
These fields repeat for each non-switched controller		CHAR(10)	Attached nonswitched controller name
		CHAR(2)	Reserved
These fields repeat for each switched controller		CHAR(10)	Switched controller name
		CHAR(2)	Reserved
These fields repeat for each active switched controller		CHAR(10)	Active switched controller name
		CHAR(2)	Reserved

LIND0500 Format: This format returns detailed information about a line of category *ETH.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
112	70	BINARY(4)	Maximum controllers
116	74	BINARY(4)	Link speed
120	78	BINARY(4)	Cost per connect time
124	7C	BINARY(4)	Cost per byte
128	80	BINARY(4)	User-defined 1
132	84	BINARY(4)	User-defined 2
136	88	BINARY(4)	User-defined 3
140	8C	BINARY(4)	Autodelete controller
144	90	BINARY(4)	Recovery limits: count limit
148	94	BINARY(4)	Recovery limits: time interval
152	98	BINARY(4)	Offset to list of active switched controllers
156	9C	BINARY(4)	Number of active switched controllers
160	A0	BINARY(4)	Entry length for list of active switched controllers
164	A4	BINARY(4)	Offset to list of SSAPs
168	A8	BINARY(4)	Number of SSAPs
172	AC	BINARY(4)	Entry length for list of SSAPs
176	B0	BINARY(4)	Offset to list of group addresses
180	B4	BINARY(4)	Number of group addresses
184	B8	BINARY(4)	Entry length for list of group addresses
188	BC	CHAR(10)	Resource name
198	C6	CHAR(10)	Network controller
208	D0	CHAR(12)	Local adapter address
220	DC	CHAR(10)	Exchange identifier
230	E6	CHAR(10)	Ethernet standard
240	F0	CHAR(10)	Error threshold level

Offset		Type	Field
Dec	Hex		
250	FA	CHAR(10)	Security for line
260	104	CHAR(10)	Propagation delay
270	10E	CHAR(10)	Autocreate controller
These fields repeat for each active switched controller		CHAR(10)	Active switched controller name
		CHAR(2)	Reserved
These fields repeat for each SSAP		BINARY(4)	SSAP maximum frame
		CHAR(10)	SSAP address
		CHAR(10)	SSAP type
This field repeats for each group address		CHAR(12)	Group address

LIND0600 Format: This format returns detailed information about a line of category *IDLC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
112	70	BINARY(4)	Line speed
116	74	BINARY(4)	CRC errors received
120	78	BINARY(4)	Short frame
124	7C	BINARY(4)	Receive overrun
128	80	BINARY(4)	Transmit underrun
132	84	BINARY(4)	Frame aborts
136	88	BINARY(4)	Retransmitted frames
140	8C	BINARY(4)	Frame sequence errors
144	90	BINARY(4)	Maximum frame size
148	94	BINARY(4)	Default window size
152	98	BINARY(4)	Frame retry limit
156	9C	BINARY(4)	Response timer
160	A0	BINARY(4)	Connect retry count
164	A4	BINARY(4)	Link speed
168	A8	BINARY(4)	Cost per connect time
172	AC	BINARY(4)	Cost per byte
176	B0	BINARY(4)	User-defined 1
180	B4	BINARY(4)	User-defined 2
184	B8	BINARY(4)	User-defined 3
188	BC	BINARY(4)	Recovery limits: count limit
192	C0	BINARY(4)	Recovery limits: time interval
196	C4	BINARY(4)	Offset to list of attached nonswitched controllers
200	C8	BINARY(4)	Entry length for list of attached nonswitched controllers
204	CC	BINARY(4)	Offset to list of active switched controllers
208	D0	BINARY(4)	Number of active switched controllers
212	D4	BINARY(4)	Entry length for list of active switched controllers
216	D8	BINARY(4)	Offset to list of switched NWIs
220	DC	BINARY(4)	Number of switched NWIs
224	E0	BINARY(4)	Entry length for list of switched NWIs
228	E4	CHAR(10)	Connection type
238	EE	CHAR(10)	Attached nonswitched NWI
248	F8	CHAR(10)	NWI channel type
258	102	CHAR(10)	NWI channel number
268	10C	CHAR(10)	Switched connection type
278	116	CHAR(10)	Connection list
288	120	CHAR(10)	Exchange identifier
298	12A	CHAR(10)	Error threshold level
308	134	CHAR(13)	Information transfer type
321	141	CHAR(10	Switched NWI selection
331	14B	CHAR(10)	Security for line
341	155	CHAR(10)	Propagation delay
These fields repeat for each non-switched controller		CHAR(10)	Attached nonswitched controller name
		CHAR(2)	Reserved
These fields repeat for each active switched controller		CHAR(10)	Active switched controller name
		CHAR(2)	Reserved
These fields repeat for each switched NWI		CHAR(10)	NWI name
		CHAR(10)	NWI channel type
		CHAR(10)	NWI channel number
		CHAR(2)	Reserved

LIND0700 Format: This format returns detailed information about a line of category *NET.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Offset to list of attached nonswitched controllers
112	70	BINARY(4)	Entry length for list of attached nonswitched controllers
116	74	CHAR(10)	Attached nonswitched NWI
These fields repeat for each non-switched controller		CHAR(10)	Attached nonswitched controller name
		CHAR(2)	Reserved

LIND0800 Format: This format returns detailed information about a line of category *SDLC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
112	70	BINARY(4)	Maximum controllers
116	74	BINARY(4)	Line speed
120	78	BINARY(4)	SHM call timer
124	7C	BINARY(4)	SHM maximum connect timer
128	80	BINARY(4)	SHM answer delay timer
132	84	BINARY(4)	Connect poll retry
136	88	BINARY(4)	Connect timer
140	8C	BINARY(4)	Short timer
144	90	BINARY(4)	Long timer
148	94	BINARY(4)	Short retry
152	98	BINARY(4)	Long retry
156	9C	BINARY(4)	Maximum frame size
160	A0	BINARY(4)	Maximum outstanding frames
164	A4	BINARY(4)	Inactivity timer
168	A8	BINARY(4)	Poll response delay
172	AC	BINARY(4)	Nonproductive receive timer
176	B0	BINARY(4)	Idle timer
180	B4	BINARY(4)	Connect poll timer
184	B8	BINARY(4)	Poll cycle pause
188	BC	BINARY(4)	Frame retry
192	C0	BINARY(4)	Fair polling timer
196	C4	BINARY(4)	Data Set Ready drop timer
200	C8	BINARY(4)	Clear To Send timer
204	CC	BINARY(4)	Remote answer timer
208	D0	BINARY(4)	Link speed
212	D4	BINARY(4)	Cost per connect time
216	D8	BINARY(4)	Cost per byte
220	DC	BINARY(4)	User-defined 1
224	E0	BINARY(4)	User-defined 2
228	E4	BINARY(4)	User-defined 3
232	E8	BINARY(4)	Recovery limits: count limit
236	EC	BINARY(4)	Recovery limits: time interval
240	F0	BINARY(4)	Offset to list of attached nonswitched controllers
244	F4	BINARY(4)	Entry length for list of attached nonswitched controllers
248	F8	BINARY(4)	Offset to list of active switched controllers
252	FC	BINARY(4)	Number of active switched controllers
256	100	BINARY(4)	Entry length for list of active switched controllers
260	104	BINARY(4)	Offset to list of resource names
264	108	BINARY(4)	Number of resource names
268	10C	BINARY(4)	Entry length for list of resource names
272	110	BINARY(4)	Offset to list of call progress signal retry values
276	114	BINARY(4)	Number of call progress signal retry values
280	118	BINARY(4)	Entry length for list of call progress signal retry values
284	11C	CHAR(10)	Data link role
294	126	CHAR(10)	Physical interface
304	130	CHAR(10)	Connection type
314	13A	CHAR(10)	Switched network backup
324	144	CHAR(10)	Activate switched network backup
334	14E	CHAR(10)	SHM node type
344	158	CHAR(10)	Autocall unit
354	162	CHAR(10)	Exchange identifier
364	16C	CHAR(10)	NRZI data encoding

Offset		Type	Field
Dec	Hex		
374	176	CHAR(10)	Clocking
384	180	CHAR(10)	Modem type supported
394	18A	CHAR(10)	Modem data rate select
404	194	CHAR(10)	Switched connection type
414	19E	CHAR(10)	Autoanswer
424	1A8	CHAR(10)	Autodial
434	1B2	CHAR(10)	Dial command type
444	1BC	CHAR(10)	Autocall resource name
454	1C6	CHAR(10)	SHM call format
464	1D0	CHAR(10)	SHM access code
474	1DA	CHAR(32)	Calling number
506	1FA	CHAR(10)	Station address
516	204	CHAR(10)	Error threshold level
526	20E	CHAR(10)	Duplex
536	218	CHAR(10)	Modulus
546	222	CHAR(10)	Autoanswer type
556	22C	CHAR(10)	Security for line
566	236	CHAR(10)	Propagation delay
These fields repeat for each non-switched controller		CHAR(10)	Attached nonswitched controller name
		CHAR(2)	Reserved
These fields repeat for each active switched controller		CHAR(10)	Active switched controller name
		CHAR(2)	Reserved
These fields repeat for each resource name		CHAR(10)	Resource name
		CHAR(2)	Reserved
These fields repeat for each call signal retry value		CHAR(10)	Call progress signal retry value
		CHAR(2)	Reserved

LIND0900 Format: This format returns detailed information about a line of category *TDLC.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Offset to list of attached nonswitched controllers
112	70	BINARY(4)	Entry length for list of attached nonswitched controllers
116	74	CHAR(10)	Attached work station controller
These fields repeat for each non-switched controller		CHAR(10)	Attached nonswitched controller name
		CHAR(2)	Reserved

LIND1000 Format: This format returns detailed information about a line of category *TRN.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
112	70	BINARY(4)	Maximum controllers
116	74	BINARY(4)	Line speed
120	78	BINARY(4)	Maximum frame size
124	7C	BINARY(4)	Link speed
128	80	BINARY(4)	Cost per connect time
132	84	BINARY(4)	Cost per byte
136	88	BINARY(4)	User-defined 1
140	8C	BINARY(4)	User-defined 2
144	90	BINARY(4)	User-defined 3
148	94	BINARY(4)	Autodelete controller
152	98	BINARY(4)	Recovery limits: count limit
156	9C	BINARY(4)	Recovery limits: time interval
160	A0	BINARY(4)	Offset to list of active switched controllers
164	A4	BINARY(4)	Number of active switched controllers
168	A8	BINARY(4)	Entry length for list of active switched controllers
172	AC	BINARY(4)	Offset to list of SSAPs
176	B0	BINARY(4)	Number of SSAPs
180	B4	BINARY(4)	Entry length for list of SSAPs
184	B8	BINARY(4)	Offset to list of function addresses
188	BC	BINARY(4)	Number of function addresses
192	C0	BINARY(4)	Entry length for list of function addresses
196	C4	CHAR(10)	Resource name
206	CE	CHAR(10)	Network controller

Offset		Type	Field
Dec	Hex		
216	D8	CHAR(10)	TRLAN manager logging level: configured
226	E2	CHAR(10)	TRLAN manager logging level: current
236	EC	CHAR(12)	TRLAN manager mode
248	F8	CHAR(10)	Log configuration changes
258	102	CHAR(10)	Token-ring inform of beacon
268	10C	CHAR(12)	Local adapter address
280	118	CHAR(10)	Exchange identifier
290	122	CHAR(10)	Early token release
300	12C	CHAR(10)	Error threshold level
310	136	CHAR(10)	Security for line
320	140	CHAR(10)	Propagation delay
330	14A	CHAR(10)	Autocreate controller
These fields repeat for each active switched controller		CHAR(10)	Active switched controller name
		CHAR(2)	Reserved
These fields repeat for each SSAP		BINARY(4)	SSAP maximum frame
		CHAR(10)	SSAP address
		CHAR(10)	SSAP type
This field repeats for each function address		CHAR(12)	Function address

LIND1100 Format: This format returns detailed information about a line of category *X25.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
112	70	BINARY(4)	Line speed
116	74	BINARY(4)	Maximum frame size
120	78	BINARY(4)	Default packet size: transmit
124	7C	BINARY(4)	Default packet size: receive
128	80	BINARY(4)	Maximum packet size: transmit
132	84	BINARY(4)	Maximum packet size: receive
136	88	BINARY(4)	Default window size: transmit
140	8C	BINARY(4)	Default window size: receive
144	90	BINARY(4)	Idle timer
148	94	BINARY(4)	Frame retry
152	98	BINARY(4)	Predial delay
156	9C	BINARY(4)	Redial delay
160	A0	BINARY(4)	Dial retries
164	A4	BINARY(4)	Switched disconnect timers: minimum connection
168	A8	BINARY(4)	Switched disconnect timers: disconnect delay
172	AC	BINARY(4)	Data Set Ready drop timer
176	B0	BINARY(4)	Clear To Send timer
180	B4	BINARY(4)	Remote answer timer
184	B8	BINARY(4)	Link speed
188	BC	BINARY(4)	Cost per connect time
192	C0	BINARY(4)	Cost per byte
196	C4	BINARY(4)	User-defined 1
200	C8	BINARY(4)	User-defined 2
204	CC	BINARY(4)	User-defined 3
208	D0	BINARY(4)	Recovery limits: count limit
212	D4	BINARY(4)	Recovery limits: time interval
216	D8	BINARY(4)	Offset to list of switched controllers
220	DC	BINARY(4)	Number of switched controllers
224	E0	BINARY(4)	Entry length for list of switched controllers
228	E4	BINARY(4)	Offset to list of active switched controllers
232	E8	BINARY(4)	Number of active switched controllers
236	EC	BINARY(4)	Entry length for list of active switched controllers
240	F0	BINARY(4)	Offset to list of logical channel entries
244	F4	BINARY(4)	Number of logical channel entries
248	F8	BINARY(4)	Entry length for list of logical channel entries
252	FC	BINARY(4)	Offset to list of switched NWIs
256	100	BINARY(4)	Number of switched NWIs
260	104	BINARY(4)	Entry length for list of switched NWIs

Offset		Type	Field
Dec	Hex		
264	108	CHAR(10)	Resource name
274	112	CHAR(20)	Local network address
294	126	CHAR(10)	Connection initiation
304	130	CHAR(10)	Physical interface
314	13A	CHAR(10)	Connection type
324	144	CHAR(10)	Attached nonswitched NWI
334	14E	CHAR(10)	NWI channel type
344	158	CHAR(10)	NWI channel number
354	162	CHAR(10)	X.25 DCE support
364	16C	CHAR(10)	Network controller
374	176	CHAR(10)	Exchange identifier
384	180	CHAR(10)	Packet mode
394	18A	CHAR(13)	Information transfer type
407	197	CHAR(10)	Extended network addressing
417	1A1	CHAR(10)	Modulus
427	1AB	CHAR(10)	Insert network address in packets
437	1B5	CHAR(10)	Error threshold level
447	1BF	CHAR(32)	Connection number
479	1DF	CHAR(32)	Calling number
511	1FF	CHAR(10)	Modem type supported
521	209	CHAR(10)	Modem data rate select
531	213	CHAR(10)	Switched connection type
541	21D	CHAR(10)	Outgoing connection list
551	227	CHAR(10)	Outgoing connection list entry
561	231	CHAR(10)	Incoming connection list
571	23B	CHAR(10)	Autoanswer
581	245	CHAR(10)	Autodial
591	24F	CHAR(10)	Dial command type
601	259	CHAR(10)	Call immediate
611	263	CHAR(10)	Autocall unit
621	26D	CHAR(10)	Autocall resource name
631	277	CHAR(10)	Switched disconnect
641	281	CHAR(10)	Autoanswer type
651	28B	CHAR(10)	Switched NWI selection
661	295	CHAR(10)	Security for line
671	29F	CHAR(10)	Propagation delay
681	2A9	CHAR(214)	Network user identification facility
These fields repeat for each switched controller		CHAR(10)	Switched controller name
		CHAR(2)	Reserved
These fields repeat for each active switched controller		CHAR(10)	Active switched controller name
		CHAR(2)	Reserved
These fields repeat for each logical channel entry		CHAR(10)	Logical channel identifier
		CHAR(10)	Logical channel type
		CHAR(10)	Logical channel controller
		CHAR(2)	Reserved
These fields repeat for each switched NWI		CHAR(10)	NWI name
		CHAR(10)	NWI channel type
		CHAR(10)	NWI channel number
		CHAR(2)	Reserved

LIND1200 Format: This format returns detailed information about a line of category *DDI.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
112	70	BINARY(4)	Maximum controllers
116	74	BINARY(4)	Maximum frame size
120	78	BINARY(4)	Token rotation time
124	7C	BINARY(4)	Link speed
128	80	BINARY(4)	Autodelete controller
132	84	BINARY(4)	Cost per connect time
136	88	BINARY(4)	Cost per byte
140	8C	BINARY(4)	User-defined 1
144	90	BINARY(4)	User-defined 2
148	94	BINARY(4)	User-defined 3
152	98	BINARY(4)	Recovery limits: count limit
156	9C	BINARY(4)	Recovery limits: time interval
160	A0	BINARY(4)	Offset to list of active switched controllers
164	A4	BINARY(4)	Number of active switched controllers
168	A8	BINARY(4)	Entry length for list of active switched controllers
172	AC	BINARY(4)	Offset to list of SSAPs
176	B0	BINARY(4)	Number of SSAPs
180	B4	BINARY(4)	Entry length for list of SSAPs
184	B8	BINARY(4)	Offset to list of group addresses

Offset		Type	Field
Dec	Hex		
188	BC	BINARY(4)	Number of group addresses
192	C0	BINARY(4)	Entry length for list of group addresses
196	C4	CHAR(10)	Resource name
206	CE	CHAR(10)	NWI name
216	D8	CHAR(10)	Network interface DLC identifier
226	E2	CHAR(12)	Local adapter address
238	EE	CHAR(10)	Exchange identifier
248	F8	CHAR(10)	Attach mode
258	102	CHAR(10)	Security for line
268	10C	CHAR(10)	Propagation delay
278	116	CHAR(10)	Autocreate controller
These fields repeat for each active switched controller		CHAR(10)	Active switched controller name
		CHAR(2)	Reserved
These fields repeat for each SSAP		BINARY(4)	SSAP maximum frame
		CHAR(10)	SSAP address
		CHAR(10)	SSAP type
This field repeats for each group address		CHAR(12)	Group address

LIND1300 Format: This format returns detailed information about a line of category *FR.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
112	70	BINARY(4)	Maximum controllers
116	74	BINARY(4)	Maximum frame size
120	78	BINARY(4)	Link speed
124	7C	BINARY(4)	Cost per connect time
128	80	BINARY(4)	Cost per byte
132	84	BINARY(4)	Network interface DLC identifier
136	88	BINARY(4)	User-defined 1
140	8C	BINARY(4)	User-defined 2
144	90	BINARY(4)	User-defined 3
148	94	BINARY(4)	Recovery limits: count limit
152	98	BINARY(4)	Recovery limits: time interval
156	9C	BINARY(4)	Offset to list of active switched controllers
160	A0	BINARY(4)	Number of active switched controllers
164	A4	BINARY(4)	Entry length for list of active switched controllers
168	A8	BINARY(4)	Offset to list of SSAPs
172	AC	BINARY(4)	Number of SSAPs
176	B0	BINARY(4)	Entry length for list of SSAPs
180	B4	CHAR(10)	Attached nonswitched NWI
190	BE	CHAR(10)	Exchange identifier
200	C8	CHAR(10)	Security for line
210	D2	CHAR(10)	Propagation delay
These fields repeat for each active switched controller		CHAR(10)	Active switched controller name
		CHAR(2)	Reserved
These fields repeat for each SSAP		BINARY(4)	SSAP maximum frame
		CHAR(10)	SSAP address
		CHAR(10)	SSAP type

LIND1400 Format: This format returns detailed information about a line of category *FAX.

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format LIND0100
108	6C	BINARY(4)	Vary on wait
120	78	BINARY(4)	Offset to list of attached nonswitched controllers
124	7C	BINARY(4)	Entry length for list of attached nonswitched controllers
128	80	BINARY(4)	Offset to list of resource names
132	84	BINARY(4)	Number of resource names
136	88	BINARY(4)	Entry length for list of resource names
These fields repeat for each nonswitched controller		CHAR(10)	Attached nonswitched controller name
		CHAR(2)	Reserved

Offset		Type	Field
Dec	Hex		
These fields repeat for each resource name		CHAR(10)	Resource name
		CHAR(2)	Reserved

Field Descriptions

Fields in the various formats returned by this API are described in the chapter on communications line descriptions in the *OS/400* Communications Configuration Reference*.

In certain cases, numeric values are assigned by this API to represent character values for some of the returned fields. Where a numeric value is assigned, the numeric value and the equivalent character value are listed as an *Exception* in the following field descriptions.

Activate switched network backup (ACTSNBU). Whether the switched network backup is activated.

Active switched controller name. The name of a controller associated with this line.

Application type (APPTYPE). The type of application that this BSC line is used for. (See the APPTYPE keyword in the *OS/400* Communications Configuration Reference*.)

Attach mode. The attach mode specified when the line was created.

Attached nonswitched controller name. The name of a controller associated with this line.

Attached nonswitched NWI. The name of the nonswitched network interface (NWI) description that contains the channel to which this line is to be attached. (See the NWI keyword in the *OS/400* Communications Configuration Reference*.)

Attached workstation controller (WSC). The name of the controller description for the 5394 work station controller or the work station controller to which the personal computer is attached. (See the WSC keyword in the *OS/400* Communications Configuration Reference*.)

Autoanswer (AUTOANS). Whether you intend to use your modem's automatic answer function. (See the AUTOANS keyword in the *OS/400* Communications Configuration Reference*.)

Autoanswer type (AUTOANSTYP). The method to be used by the system and modem to answer incoming calls. (See the AUTOANSTYP keyword in the *OS/400* Communications Configuration Reference*.)

Autocall resource name (ACRSRCNAME). The name that is assigned by the system to a communications port from which a communications line attaches to an automatic call unit. (See the ACRSRCNAME keyword in the *OS/400* Communications Configuration Reference*.)

Autocall unit (AUTOCALL). An associated automatic call unit. (See the AUTOCALL keyword in the *OS/400* Communications Configuration Reference*.)

Autocreate controller (AUTOCRTCTL). Whether the system is to automatically create APPC controller descriptions when incoming calls are received from other systems on the local area network. (See the AUTOCRTCTL keyword in the *OS/400* Communications Configuration Reference*.)

Autodelete controller (AUTODLTCTL). The number of minutes the system should wait before automatically varying off and deleting automatically created controller descriptions associated with this line. (See the AUTODLTCTL keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -3 implies *NONE

Autodial (AUTODIAL). Whether or not you intend to use the automatic call function to dial the remote system or network to establish a switched line connection. (See the AUTODIAL keyword in the *OS/400* Communications Configuration Reference*.)

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 was this large or larger.

Bytes returned. The amount of data returned to the calling program in the receiver variable.

Call immediate (CALLIMMED). For switched X.25 lines, whether a call should be made immediately after varying on the line description. (See the CALLIMMED keyword in the *OS/400* Communications Configuration Reference*.)

Calling number (CALLNBR). The local connection number of a line. (See the CALLNBR keyword in the *OS/400* Communications Configuration Reference*.)

Call progress signal retry value (CPSRTY). Whether a call attempt should be retried if the specified call progress signal is received. (See the CPSRTY keyword in the *OS/400* Communications Configuration Reference*.)

Character code (CODE). The type of character code used. (See the CODE keyword in the *OS/400* Communications Configuration Reference*.)

Clear to send timer (CTSTMR). The length of time that the system should wait for the modem (DCE) to raise or drop Clear to Send (CTS) before signaling an error. (See the CTSTMR keyword in the *OS/400* Communications Configuration Reference*.)

| **Clocking (CLOCK)**. Specifies that the clocking function for the line is provided by the modem (*MODEM). (See the CLOCK keyword in the *OS/400* Communications Configuration Reference*.)

| **Connection initiation (CNNINIT)**. The values to initiate the X.25 data link connection. (See the CNNINIT keyword in the *OS/400* Communications Configuration Reference*.)

| **Connection list (CNNLSTIN)**. The name of the connection list used to retrieve ISDN call information when authorizing incoming calls. (See the CNNLSTIN keyword in the *OS/400* Communications Configuration Reference*.)

| **Connection number (CNNNBR)**. For switched X.25 lines, the number of the remote DCE (packet switched data network) that can be contacted using this line description. (See the CNNNBR keyword in the *OS/400* Communications Configuration Reference*.)

| **Connect poll retry (CNNPOLLRTY)**. The number of connect poll retries that will be attempted before the AS/400 system indicates an error in contacting the remote system or controller. (See the CNNPOLLRTY keyword in the *OS/400* Communications Configuration Reference*.)

| **Connect poll timer (CNNPOLLTMR)**. The length of time that the system waits for the response to a poll while in normal disconnect mode before polling the next station. (See the CNNPOLLTMR keyword in the *OS/400* Communications Configuration Reference*.)

| **Connect retry count (IDLCNNRTY)**. The number of times to retry a transmission at connection time. (See the IDLCCNNRTY keyword in the *OS/400* Communications Configuration Reference*.)

| *Exceptions:*

| • Value of -8 implies *NOMAX
| • Value of -9 implies *CNN

| **Connect timer (CNNTMR)**. The amount of time that an automatic answer connect request waits for an incoming call on an X.21 circuit-switched line. (See the CNNTMR keyword in the *OS/400* Communications Configuration Reference*.)

| *Exception:*

| • Value of -8 implies *NOMAX

| **Connection type (CNN)**. The type of line connection. (See the CNN keyword in the *OS/400* Communications Configuration Reference*.)

| **Contention state retry (CTNRTY)**. The number of contention state retries that can be attempted before disconnecting the line. (See the CTNRTY keyword in the *OS/400* Communications Configuration Reference*.)

| **Continue timer (CONTTMR)**. The length of time that the system waits before sending a TTD or WACK control character. (See the CONTTMR keyword in the *OS/400* Communications Configuration Reference*.)

| **Controller category**. This value will be one of the following:

| *APPC
| *ASC
| *BSC
| *FNC
| *HOST
| *LWS
| *NET
| *RTL
| *RWS
| *VWS
| *TAP

| The category value is derived from the command used to create the controller description.

| **Controller name (CTL)**. The names of one or more controller descriptions associated with this line.

| **Controller text description**. A brief description of a controller associated with this line.

| **Controller type**. The type of controller being described. See the TYPE keyword in the *OS/400* Communications Configuration Reference* for a full list of the possible type values.

| **Cost per byte (COSTBYTE)**. The relative cost per byte of sending and receiving data on the line. (See the COSTBYTE keyword in the *OS/400* Communications Configuration Reference*.)

| *Exception:*

| • Value of -9 implies *CNN

| **Cost per connect time (COSTCNN)**. The relative cost of being connected on the line. (See the COSTCNN keyword in the *OS/400* Communications Configuration Reference*.)

| *Exception:*

| • Value of -9 implies *CNN

| **CRC errors received (CRCRCV)**. The level of error threshold monitoring done by the system for CRC errors received. (See the CRCRCV keyword in the *OS/400* Communications Configuration Reference*.)

| *Exceptions:*

| • Value of -4 implies *OFF
| • Value of -5 implies *MIN
| • Value of -6 implies *MED
| • Value of -7 implies *MAX

| **Data bits per character (BITSCHAR)**. The data bits per character (either 7 or 8 bits excluding the parity bit). (See the BITSCHAR keyword in the *OS/400* Communications Configuration Reference*.)

| **Data link role (ROLE)**. Whether this system is the primary station or secondary station, or if this station should dynamically negotiate its role with the remote station when the line

is varied on. (See the ROLE keyword in the *OS/400* Communications Configuration Reference*.)

Data set ready drop timer (DSRDRPTMR). The length of time that the system should wait for the modem (DCE) to drop Data Set Ready (DSR) after the system has dropped Data Terminal Ready (DTR) before signalling an error. (See the DSRDRPTMR keyword in the *OS/400* Communications Configuration Reference*.)

Data state retry (DTASTTRTY). The time before retry when BSC is sending or receiving data on the communications line. (See the DTASTTRTY keyword in the *OS/400* Communications Configuration Reference*.)

Date information retrieved. The date that the information was provided by the API. This is returned as 7 characters in the form CYYMMDD, where:

C Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
YY Year
MM Month
DD Day

Default packet size (DFTPKTSIZE). The default packet size to use for controllers attached to this line description. This information is returned in two separate fields:

- *Transmit*
- *Receive*

(See the DFTPKTSIZE keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -10 implies *TRANSMIT

Default window size: transmit/receive (IDLCWDWSIZ, DFTWDWSIZE). The default window size used for this line description. (See the IDLCWDWSIZ and DFTWDWSIZE keywords in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -9 implies *CNN
- Value of -10 implies *TRANSMIT

Dial command type (DIALCMD). The dial command type used to establish a connection with a remote system. (See the DIALCMD keyword in the *OS/400* Communications Configuration Reference*.)

Dial retries (DIALRTY). The number of times to retry dialing the number before considering the dialing unsuccessful. (See the DIALRTY keyword in the *OS/400* Communications Configuration Reference*.)

Duplex (DUPLEX). Whether the AS/400 system will leave the request-to-send (RTS) modem signal on continuously, or whether the RTS will be raised when the AS/400 system must transmit data and dropped when it is finished transmitting. (See the DUPLEX keyword in the *OS/400* Communications Configuration Reference*.)

Early token release (ELYTKNRLS). Allows greater throughput on 16MB token-ring network lines. (See the ELYTKNRLS keyword in the *OS/400* Communications Configuration Reference*.)

Echo support (ECHO). Whether the AS/400 system is to send back (echo) to the remote station none of the characters that it receives, all of the characters it receives, or all data up to, but not including, the end-of-record character (*CNTL). (See the ECHO keyword in the *OS/400* Communications Configuration Reference*.)

Entry length for list of active switched controllers. The entry length in bytes of each element in the list of active switched controllers returned with this format. A value of zero is returned if the list is empty.

Entry length for list of attached nonswitched controllers. The entry length in bytes of each element in the list of attached nonswitched controllers returned with this format. A value of zero is returned if the list is empty.

Entry length for list of call progress signal retry values. The entry length in bytes of each element in the list of call progress signal retry values returned with this format. A value of zero is returned if the list is empty.

Entry length for list of EOR characters. The entry length in bytes of each element in the list of end-of-record (EOR) characters returned with this format. A value of zero is returned if the list is empty.

Entry length for list of function addresses. The entry length in bytes of each element in the list of function addresses returned with this format. A value of zero is returned if the list is empty.

Entry length for list of group addresses. The entry length in bytes of each element in the list of group addresses returned with this format. A value of zero is returned if the list is empty.

Entry length for list of logical channel entries. The entry length in bytes of each element in the list of logical channel entries returned with this format. A value of zero is returned if the list is empty.

Entry length for list of resource names. The entry length in bytes of each element in the list of resource names returned with this format. A value of zero is returned if the list is empty.

Entry length for list of SSAPs. The entry length in bytes of each element in the list of SSAPs returned with this format. A value of zero is returned if the list is empty.

Entry length for list of switched controllers. The entry length in bytes of each element in the list of switched control-

lers returned with this format. A value of zero is returned if the list is empty.

Entry length for list of switched NWIs. The entry length in bytes of each element in the list of switched network interface (NWI) descriptions returned with this format. A value of zero is returned if the list is empty.

EOR character. The end-of-record character.

Error threshold level (THRESHOLD). The level of the error threshold that is monitored by the system. (See the THRESHOLD keyword in the *OS/400* Communications Configuration Reference*.)

Ethernet standard (ETHSTD). The standard used by the Ethernet local area network. (See the ETHSTD keyword in the *OS/400* Communications Configuration Reference*.)

Exchange identifier (EXCHID). The exchange identifier that the local AS/400 system can send to the remote controller or system. (See the EXCHID keyword in the *OS/400* Communications Configuration Reference*.)

Extended network addressing (EXNNETADR). Whether extended network addressing is used by this line description and attached controller descriptions. (See the EXNNETADR keyword in the *OS/400* Communications Configuration Reference*.)

Fair polling timer (FAIRPLLTMR). The maximum length of time (in seconds) that the system will send data to one or more stations on the line before polling stations without pending output requests. (See the FAIRPLLTMR keyword in the *OS/400* Communications Configuration Reference*.)

Flow control (FLOWCNTL). Whether you will use the XON and XOFF flow control characters to control the flow of data to your system. (See the FLOWCNTL keyword in the *OS/400* Communications Configuration Reference*.)

Frame aborts (ABORTS). The level of error threshold monitoring done by the system for frame aborts. (See the ABORTS keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -4 implies *OFF
- Value of -5 implies *MIN
- Value of -6 implies *MED
- Value of -7 implies *MAX

Frame retry (FRAMERTY). The number of retries for an unanswered or unacknowledged frame. (See the FRAMERTY keyword in the *OS/400* Communications Configuration Reference*.)

Frame retry limit (IDLCFRMRTY). The maximum number of frame retransmissions to attempt before initiating recovery. (See the IDLCFRMRTY keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -9 implies *CNN

Frame sequence errors (FRMSEQERR). The level of error threshold monitoring done by the system for frame sequence errors. (See the FRMSEQERR keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -4 implies *OFF
- Value of -5 implies *MIN
- Value of -6 implies *MED
- Value of -7 implies *MAX

Function address. Functional address used by token-ring network lines.

Group address. The group of addresses to which a subset of nodes on the network respond to, in addition to their local adapter addresses.

Idle timer (IDLTMR). The maximum allowable time between characters before the adapter forwards the receive buffer to the system. (See the IDLTMR keyword in the *OS/400* Communications Configuration Reference*.)

Inactivity timer (INACTTMR). The time that the system waits for activity on a line before disconnecting. (See the INACTTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -8 implies *NOMAX

Include STX character in the LRC (STXLRC). Whether to exclude the start-of-text (STX) character in the longitudinal redundancy check (LRC) calculation. (See the STXLRC keyword in the *OS/400* Communications Configuration Reference*.)

Incoming connection list (CNNLSTIN). The name of the connection list used to retrieve ISDN call information when authorizing incoming calls. (See the CNNLSTIN keyword in the *OS/400* Communications Configuration Reference*.)

Information transfer type (INFTRFTYPE). How data is to be encoded for the ISDN B-channel associated with this line description. (See the INFTRFTYPE keyword in the *OS/400* Communications Configuration Reference*.)

Insert network address in packets (ADRINSERT). Whether the AS/400 system inserts the local network address (NETADR) in call-request and call-accept packets. (See the ADRINSERT keyword in the *OS/400* Communications Configuration Reference*.)

Line category. This value will be one of the following:

- *ASC
- *BSC
- *DDI
- *ETH
- *FAX

| *FR
| *IDLC
| *NET
| *SDLC
| *TDLC
| *TRN
| *X25

| The category value is derived from the command used to
| create the line description.

| **Line name**. The name of the line description.

| **Line speed (LINESPEED)**. The line speed in bits per
| second (bps). (See the LINESPEED keyword in the *OS/400*
| *Communications Configuration Reference*.)

| *Exception:*

| • Value of -11 implies *CALC

| **Link speed (LINKSPEED)**. The link speed in bits per
| second (bps). (See the LINKSPEED keyword in the *OS/400*
| *Communications Configuration Reference*.)

| *Exceptions:*

| • Value of -5 implies *MIN
| • Value of -7 implies *MAX
| • Value of -12 implies *INTERFACE
| • Value of -23 implies 10 megabits per second
| • Value of -24 implies 4 megabits per second
| • Value of -25 implies 16 megabits per second

| **Local adapter address**. The address used by the adapter
| for this line to transmit from and answer to on the token-ring
| or LAN. (See the ADPTADR keyword in the *OS/400* Com-
| *munications Configuration Reference*.)

| **Local network address**. The network address of the
| AS/400 system. (See the NETADR keyword in the *OS/400*
| *Communications Configuration Reference*.)

| **Log configuration changes (LOGCFGCHG)**. Whether the
| token-ring network manager for this line is to log configura-
| tion changes on the ring. (See the LOGCFGCHG keyword in
| the *OS/400* Communications Configuration Reference*.)

| **Logical channel controller**. Either an SVC or an attached
| PVC controller.

| For a switched virtual circuit (SVC) logical channel, this is the
| controller description currently active on this logical channel.

| For a permanent virtual circuit (PVC) logical channel, this is
| the controller description permanently attached to this logical
| channel.

| **Logical channel identifier**. An identifier consisting of 3
| hexadecimal characters that can range from hex 001 to hex
| FFF. The first character represents the logical channel
| group, as assigned by the network subscription. The last 2
| characters are the network-assigned logical channel number.

| **Logical channel type**. The type of logical channel.

| **Long retry (LONGRTY)**. The number of bursts of call retry
| attempts that the system makes when processing a con-
| nection request. (See the LONGRTY keyword in the
| *OS/400* Communications Configuration Reference*.)

| **Long timer (LONGTMR)**. The length of time that the
| system waits between connection retry attempts. (See the
| LONGTMR keyword in the *OS/400* Communications Config-
| *uration Reference*.)

| **Maximum buffer size (MAXBUFFER)**. The maximum size
| of the incoming and outgoing buffers. (See the
| MAXBUFFER keyword in the *OS/400* Communications Con-
| *figuration Reference*.)

| **Maximum controllers (MAXCTL)**. The maximum number
| of controllers this line can support. (See the MAXCTL
| keyword in the *OS/400* Communications Configuration
| *Reference*.)

| **Maximum frame size (MAXFRAME)**. The maximum frame
| size or logical link control data unit that can be transmitted
| and received on this line. (See the MAXFRAME keyword in
| the *OS/400* Communications Configuration Reference*.)

| **Maximum outstanding frames (MAXOUT)**. The maximum
| number of information frames that can be sent to a remote
| system and received from a remote system before allowing
| the receiving system to respond. (See the MAXOUT
| keyword in the *OS/400* Communications Configuration
| *Reference*.)

| **Maximum packet size (MAXPKTSIZE)**. The maximum
| packet size that can be used by any controller associated
| with this line description. This information is returned in two
| separate fields:

| • *Transmit*
| • *Receive*

| (See the MAXPKTSIZE keyword in the *OS/400* Communica-
| *tions Configuration Reference*.)

| *Exceptions:*

| • Value of -10 implies *TRANSMIT
| • Value of -13 implies *DFTPKTSIZE

| **Modem data rate select (MODEMRATE)**. Whether this
| modem is being operated at its full rated speed, or at an
| alternate or half speed. (See the MODEMRATE keyword in
| the *OS/400* Communications Configuration Reference*.)

| **Modem type supported (MODEM)**. The type of modem
| diagnostic tests to be used on the line. (See the MODEM
| keyword in the *OS/400* Communications Configuration
| *Reference*.)

| **Modulus (MODULUS)**. Whether extended sequence
| numbers are used (modulus 128) or not (modulus 8). (See
| the MODULUS keyword in the *OS/400* Communications
| *Configuration Reference*.)

Network controller (NETCTL). The name of an existing network controller description. (See the NETCTL keyword in the *OS/400* Communications Configuration Reference*.)

Network interface DLC identifier. The data link control (DLC) identifier used to connect to the network.

Network user identification facility (NETUSRID). Allows X.25 network subscribers to specify the network user identification (NUI) selection facility that is encoded in the facility field of all call request packets sent on this line. (See the NETUSRID keyword in the *OS/400* Communications Configuration Reference*.)

Nonproductive receive timer (NPRDRCVTMR). The time that the system waits for either a final frame or an idle signal while the secondary station is continuously transmitting. (See the NPRDRCVTMR keyword in the *OS/400* Communications Configuration Reference*.)

NRZI data encoding (NRZI). Whether or not the AS/400 system uses the non-return-to-zero inverted (NRZI) transmission coding method. (See the NRZI keyword in the *OS/400* Communications Configuration Reference*.)

Number of active switched controllers. The number of elements in the list of active switched controllers returned with this format. A value of zero is returned if the list is empty.

Number of attached nonswitched controllers. The number of elements in the list of attached nonswitched controllers returned with this format. A value of zero is returned if the list is empty.

Number of call progress signal retry values. The number of elements in the list of call progress signal retry values returned with this format. A value of zero is returned if the list is empty.

Number of EOR characters. The number of elements in the list of end-of-record (EOR) characters returned with this format. A value of zero is returned if the list is empty.

Number of function addresses. The number of elements in the list of function addresses returned with this format. A value of zero is returned if the list is empty.

Number of group addresses. The number of elements in the list of group addresses returned with this format. A value of zero is returned if the list is empty.

Number of logical channel entries. The number of elements in the list of logical channel entries returned with this format. A value of zero is returned if the list is empty.

Number of resource names. The number of elements in the list of resource names returned with this format. A value of zero is returned if the list is empty.

Number of SSAPs. The number of elements in the list of source service access points (SSAPs) returned with this format. A value of zero is returned if the list is empty.

Number of switched controllers. The number of elements in the list of switched controllers returned with this format. A value of zero is returned if the list is empty.

Number of switched NWIs. The number of elements in the list of switched network interface (NWI) descriptions returned with this format. A value of zero is returned if the list is empty.

Number of trailing characters. The value returned with each element in the list of EOR characters.

NWI channel number (NWICHLNBR). The network interface (NWI) channel to be used by this line description. (See the NWICHLNBR keyword in the *OS/400* Communications Configuration Reference*.)

NWI channel type (NWICHLTYPE). The type of network interface channels to be used by this line description. (See the NWICHLTYPE keyword in the *OS/400* Communications Configuration Reference*.)

NWI name. The name of the existing network interface description.

Offset to list of active switched controllers. The offset in bytes to the first element in the list of active switched controllers returned with this format. A value of zero is returned if the list is empty.

Offset to list of attached nonswitched controllers. The offset in bytes to the first element in the list of attached nonswitched controllers returned with this format. A value of zero is returned if the list is empty.

Offset to list of call progress signal retry values. The offset in bytes to the first element in the list of call progress signal retry values returned with this format. A value of zero is returned if the list is empty.

Offset to list of EOR characters. The offset in bytes to the first element in the list of EOR characters returned with this format. A value of zero is returned if the list is empty.

Offset to list of function addresses. The offset in bytes to the first element in the list of function addresses returned with this format. A value of zero is returned if the list is empty.

Offset to list of group addresses. The offset in bytes to the first element in the list of group addresses returned with this format. A value of zero is returned if the list is empty.

Offset to list of logical channel entries. The offset in bytes to the first element in the list of logical channel entries returned with this format. A value of zero is returned if the list is empty.

Offset to list of resource names. The offset in bytes to the first element in the list of resource names returned with this format. A value of zero is returned if the list is empty.

Offset to list of SSAPs. The offset in bytes to the first element in the list of source service access points (SSAPs) returned with this format. A value of zero is returned if the list is empty.

Offset to list of switched controllers. The offset in bytes to the first element in the list of switched controllers returned with this format. A value of zero is returned if the list is empty.

Offset to list of switched NWIs. The offset in bytes to the first element in the list of switched network interface (NWI) descriptions returned with this format. A value of zero is returned if the list is empty.

Online at IPL (ONLINE). Whether or not the line is varied on automatically when the system is turned on. (See the ONLINE keyword in the *OS/400* Communications Configuration Reference*.)

Outgoing connection list (CNNLSTOUT). For switched ISDN connections, the name of a connection list containing the network-assigned numbers used for outgoing calls on this controller. (See the CNNLSTOUT keyword in the *OS/400* Communications Configuration Reference*.)

Outgoing connection list entry (CNNLSTOUTE). For switched ISDN connections, the name of the connection list entry containing the network-assigned numbers used for outgoing calls on this line. (See the CNNLSTOUTE keyword in the *OS/400* Communications Configuration Reference*.)

Packet mode (PKTMODE). Allows an AS/400 system to communicate directly with another system by using the B-channel X.25 virtual circuit service integrated within an ISDN. (See the PKTMODE keyword in the *OS/400* Communications Configuration Reference*.)

Physical interface (INTERFACE). The type of physical communications line interface that this communications adapter port and cable will be attached to. (See the INTERFACE keyword in the *OS/400* Communications Configuration Reference*.)

Poll cycle pause (POLLPAUSE). The length of time that the system waits after the last remote system in the poll list is polled before beginning another pass through the poll list. (See the POLLPAUSE keyword in the *OS/400* Communications Configuration Reference*.)

Poll response delay (POLLRSPDLY). The minimum duration of time that the system waits before it responds to a data poll if there is no information frame to transmit. (See the POLLRSPDLY keyword in the *OS/400* Communications Configuration Reference*.)

Predial delay (PREDIALDLY). The length of time to wait before dialing the number to establish a connection to the remote system or network. (See the PREDIALDLY keyword in the *OS/400* Communications Configuration Reference*.)

Propagation delay (PRPDLY). The time required for a signal to travel from one end of a link to the other end. (See the PRPDLY keyword in the *OS/400* Communications Configuration Reference*.)

Receive overrun (OVERRUN). The level of error threshold monitoring done by the system for receive overrun errors. (See the OVERRUN keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -4 implies *OFF
- Value of -5 implies *MIN
- Value of -6 implies *MED
- Value of -7 implies *MAX

Receive timer (RCVTMR). The maximum amount of time the AS/400 system waits for a response from the remote system before a time-out occurs. (See the RCVTMR keyword in the *OS/400* Communications Configuration Reference*.)

Receive TTD or WACK retry (RCVRTY). The number of times that a temporary text delay (TTD) or wait-before-transmit-positive acknowledgement (WACK) is received before the session fails. (See the RCVRTY keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -8 implies *NOMAX

Recovery limits (CMNRCYLMT). The second-level communications recovery limit for each line description. These limits are returned in two separate fields:

- *Count limit*: The number of second-level recovery attempts to be automatically performed by the system.
- *Time interval*: The length of time (in minutes) in which the specified number of second-level recoveries can be attempted.

(See the CMNRCYLMT keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -14 implies *SYSVAL

Redial delay (REDIALDLY). The length of time to wait before redialing the number to establish a connection to the remote system or network if the previous attempt was unsuccessful. (See the REDIALDLY keyword in the *OS/400* Communications Configuration Reference*.)

Remote answer timer (RMTANSTMR). The length of time that the system should wait for the modem (DCE) to raise Data Set Ready (DSR) after dialing before signaling an error. (See the RMTANSTMR keyword in the *OS/400* Communications Configuration Reference*.)

Reserved. Space included for alignment.

Response timer (IDLCRSPTMR). The length of time to wait before retransmitting a frame when an acknowledgement is not received. (See the IDLCRSPTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -9 implies *CNN

Resource name (RSRCNAME). The unique name that is assigned by the system to the physical equipment (in this case, a communications port) attached to the system. (See the RSRCNAME keyword in the *OS/400* Communications Configuration Reference*.)

Retransmitted frames (RETRANSMIT). The level of error threshold monitoring done by the system for retransmitted frames. (See the RETRANSMIT keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -4 implies *OFF
- Value of -5 implies *MIN
- Value of -6 implies *MED
- Value of -7 implies *MAX

Security for line (SECURITY). The types of security protection available on the line. (See the SECURITY keyword in the *OS/400* Communications Configuration Reference*.)

Short frame (SHORTFRAME). The level of error threshold monitoring done by the system for short frame errors. (See the SHORTFRAME keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -4 implies *OFF
- Value of -5 implies *MIN
- Value of -6 implies *MED
- Value of -7 implies *MAX

SHM access code (SHMACC). The access code used by an X.21 short-hold mode (SHM) line when calling a system on another network. (See the SHMACC keyword in the *OS/400* Communications Configuration Reference*.)

SHM answer delay timer (SHMANSDLY). The length of time the system waits for controllers to call in before attempting to call out. (See the SHMANSDLY keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -8 implies *NOMAX

SHM call format (SHMCALLFMT). The format of the network identifier used in the local system's connection number. (See the SHMCALLFMT keyword in the *OS/400* Communications Configuration Reference*.)

SHM call timer (SHMCALLTMR). The interval at which a connection is reestablished on an X.21 short-hold mode line to verify the state of the remote system if no normal data traffic occurs in the specified interval. (See the SHMCALLTMR keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -3 implies *NONE

SHM maximum connect timer (SHMMAXCNN). The length of time the system allows connection to any one controller when there are more controllers than there are available ports. (See the SHMMAXCNN keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -8 implies *NOMAX

SHM node type (SHMNODE). The physical unit type of the controllers using the X.21 short-hold mode line. (See the SHMNODE keyword in the *OS/400* Communications Configuration Reference*.)

Short retry (SHORTRTY). The number of retry attempts that the system makes during a burst of call retries. (See the SHORTRTY keyword in the *OS/400* Communications Configuration Reference*.)

Short timer (SHORTTMR). The length of time that the system waits between retry attempts when processing a connection request. (See the SHORTTMR keyword in the *OS/400* Communications Configuration Reference*.)

SSAP address. The hexadecimal logical channel address that is used to route data off the line to the proper user.

SSAP maximum frame. The largest frame size allowed on this source service access point (SSAP).

SSAP type. The type of communications used by the system on this SSAP.

Station address (STNADR). The address used by the remote control station to poll the AS/400 system. (See the STNADR keyword in the *OS/400* Communications Configuration Reference*.)

Stop bits (STOPBITS). The number of bits to be added to the end of each character to keep the local and remote ends of the line synchronized. (See the STOPBITS keyword in the *OS/400* Communications Configuration Reference*.)

Switched connection type (SWTCNN). Whether the line can be used for incoming and outgoing calls, incoming calls only, or outgoing calls only. (See the SWTCNN keyword in the *OS/400* Communications Configuration Reference*.)

Switched controller name. The name of a controller associated with this line.

Switched disconnect (SWTDSC). For switched lines (CNN(*SWTPP)), whether the line is to be dropped when no virtual circuits are active and the disconnection timers specified on the SWTDSCTMR parameter have expired. (See the

SWTDSC keyword in the *OS/400* Communications Configuration Reference*.)

Switched disconnect timers (SWTDSCTMR). The timers used for disconnecting switched lines from a network or remote system. The timer values are returned in two separate fields:

- *Minimum connection*: The minimum length of time the system keeps the connection active.
- *Disconnect delay*: The length of time the system waits before attempting to disconnect the switched connection when the line is idle and the minimum connection time has expired.

(See the SWTDSCTMR keyword in the *OS/400* Communications Configuration Reference*.)

Switched network backup (SNBU). Whether or not you want the switched network backup capability. (See the SNBU keyword in the *OS/400* Communications Configuration Reference*.)

Switched NWI selection (SWTNWISLCT). The the method used to select network interface (NWI) descriptions from the switched NWI list. (See the SWTNWISLCT keyword in the *OS/400* Communications Configuration Reference*.)

SYN characters (SYNCCHARS). The number of SYN characters used to establish and maintain synchronization and as time-fill characters in the absence of any data or other control character. (See the SYNCCHARS keyword in the *OS/400* Communications Configuration Reference*.)

Text description (TEXT). A brief description of the line and its location. (See the TEXT keyword in the *OS/400* Communications Configuration Reference*.)

Time information retrieved. The time that the information was provided by the API. It is returned as 6 characters in the form HHMMSS, where:

HH Hour
MM Minute
SS Second

Token-ring inform of beacon (TRNINFBCN). Whether the token-ring network manager for this line is to provide notification of beaconing on the ring to the system operator. (See the TRNINFBCN keyword in the *OS/400* Communications Configuration Reference*.)

Token rotation time. The token rotation time requested when the line was created.

Exception:

- Value of -11 implies *CALC

Transmit TTD or WACK retry (TMTRTY). The number of times that a temporary-text-delay (TTD) or wait-before-transmit-positive acknowledgement (WACK) control character is sent to hold up the line when the AS/400 system is not ready to respond to the remote end. (See the TMTRTY keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -8 implies *NOMAX

Transmit underrun (UNDERRUN). The level of error threshold monitoring done by the system for transmit underrun errors. (See the UNDERRUN keyword in the *OS/400* Communications Configuration Reference*.)

Exceptions:

- Value of -4 implies *OFF
- Value of -5 implies *MIN
- Value of -6 implies *MED
- Value of -7 implies *MAX

TRLAN manager logging level (TRNLOGLVL). The logging level to be used by the token-ring network manager. This information is returned in two separate fields:

- *Configured*
- *Current*

(See the TRNLOGLVL keyword in the *OS/400* Communications Configuration Reference*.)

TRLAN manager mode (TRNMGRMODE). Whether the token-ring network manager for this line is operating in observing or controlling mode. (See the TRNMGRMODE keyword in the *OS/400* Communications Configuration Reference*.)

Type of parity (PARITY). The type of parity for error checking. (See the PARITY keyword in the *OS/400* Communications Configuration Reference*.)

User-defined 1, 2, and 3 (USRDFN1, USRDFN2, USRDFN3). Used to describe any unique characteristics of the line that you want to control. (See the USRDFN1, USRDFN2, and USRDFN3 keywords in the *OS/400* Communications Configuration Reference*.)

Vary on wait (VRYWAIT). Whether the line is varied on synchronously or asynchronously. (See the VRYWAIT keyword in the *OS/400* Communications Configuration Reference*.)

Exception:

- Value of -15 implies *NOWAIT

X.25 DCE support (X25DCE). Allows an AS/400 system to communicate directly with another system without going through an X.25 network. (See the X25DCE keyword in the *OS/400* Communications Configuration Reference*.)

XOFF character (XOFFCHAR). The hexadecimal value used to tell your line to stop sending data. (See the XOFFCHAR keyword in the *OS/400* Communications Configuration Reference*.)

XON character (XONCHAR). The hexadecimal value used to tell your line to start sending data. (See the XONCHAR

keyword in the *OS/400* Communications Configuration Reference*.)

Error Messages

CPF24B4 E Severe error addressing parameter list.
CPF2625 E Not able to allocate object &1.
CPF2634 E Not authorized to object &1.
CPF26A7 E Category of object incompatible with API format.
CPF2703 E Controller description &1 not found.
CPF2704 E Line description &1 not found.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 not valid.
CPF3C24 E Length of receiver variable not valid.
CPF3CF1 E Error code parameter not valid.
CPF8104 E Controller description &4 damaged.
CPF811D E NWI description &4 damaged.
CPF8125 E Line description &4 damaged.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 4. Database File APIs

Chapter 10. 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

| **Process Extended Dynamic SQL** (QSQPRCED) processes Structured Query Language (SQL) extended dynamic statements in an SQL package object.

| **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

| **Note:** You may also want to reference the APIs that deal with the commitment control functions of the system. See Chapter 11, "Commitment Control APIs" on page 11-1.

| With the exception of QDBLDBR, QUSRMBRD, and QSQPRCED 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, QUSRMBRD, and QSQPRCED 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.

| Some of the database APIs return character values that have an associated CCSID (coded character set identifier). If the CCSID value for the job calling the API is not 65535, the character values are converted from their current CCSID to the CCSID of the job. This conversion may cause some data to be lost. The CCSID associated with the job is returned to the user. If the CCSID value for the job is 65535, no conversions are performed on the character values. The character value CCSID stored in the file object is returned to the user.

| 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. The API descriptions are presented in alphabetical order.

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 10-22.

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" on page 10-3.

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 10-3.

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."

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)	Member 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
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
85	55	CHAR(3)	Reserved
88	58	BINARY(4)	File text description CCSID

MBRL0100 List Data Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Member name used

MBRL0200 List Data Section

Offset		Type	Field
Dec	Hex		
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
96	60	BINARY(4)	Member text description CCSID

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.

File text description CCSID. The CCSID for the file text description. The job default CCSID of the current process will be used to translate the text. For more information about CCSID, see the *National Language Support Planning Guide*.

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. Description of the member found in the file.

Member text description CCSID. The CCSID for the member text description. The job default CCSID of the current process will be used to translate the text. For more information about CCSID, see the *National Language Support Planning Guide*.

Override processing. Whether overrides are to be processed. The possible values are:

0 No override processing
1 Override processing

Reserved. An ignored field.

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.
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 returned. 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 10-5, "DBRL0200 Format (Member)" on page 10-5, or "DBRL0300 Format (Record Format)" on page 10-5.

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" on page 10-6.

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" on page 10-6.

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" on page 10-6.

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" on page 10-6.

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

| CPF3CF2 E Error(s) occurred during running of &1 API.
| CPF3C21 E Format name &1 is not valid.
| CPF3C23 E Object &1 is not a database file.
| CPF326C E File name &1 not valid special value.
| CPF326D E Member name &1 not valid special value.
| CPF326E E Record format name &1 not valid special value.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 10-8.

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."

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
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
107	6B	CHAR(1)	Reserved
108	6C	BINARY(4)	Record text description CCSID
112	70	CHAR(1)	Variable length fields in format indicator
113	71	CHAR(1)	Graphic fields indicator
114	72	CHAR(1)	Date and time fields indicator
115	73	CHAR(1)	Null-capable fields indicator

FLDL0100 List Data Section

Offset		Type	Field
Dec	Hex		
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
212	D4	CHAR(10)	Internal field name
222	DE	CHAR(30)	Alternative field name
252	FC	BINARY(4)	Length of alternative field name
256	100	BINARY(4)	Number of DBCS characters
260	104	CHAR(1)	Null values allowed
261	105	CHAR(1)	Variable field indicator
262	106	CHAR(4)	Date and time format
266	10A	CHAR(1)	Date and time separator
267	10B	CHAR(1)	Variable length field indicator (overlay for MI mapping)
268	10C	BINARY(4)	Field text description CCSID
272	110	BINARY(4)	Field data CCSID
276	114	BINARY(4)	Field column headings CCSID
280	118	BINARY(4)	Field edit words CCSID

Field Descriptions

Alternative field name. The alternative name of the field the entry describes.

Field column headings CCSID.

0 There are no field column headings.
1–65,535 The CCSID for the field column headings.

Field data CCSID.

0 There is no field data.
1–65,535 The CCSID for the field data.

Field edit words CCSID.

0 There are no field edit words.
1–65,535 The CCSID for the field edit words.

Field text description CCSID.

0 There is no field text description.
1–65,535 The CCSID for the field text description.

Record text description CCSID.

0 There is no record text description.
1–65,535 The CCSID for the record text description.

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)
B Binary
D Digits only
E Either DBCS or alphanumeric
F Floating point
G Graphic data type
H Hexadecimal
I Inhibit entry

| *J* Double-byte character set (DBCS) data only
| *L* Date
| *M* Numeric only
| *N* Numeric shift
| *O* (Open) Both DBCS and alphanumeric
| *P* Packed decimal
| *S* Zoned decimal
| *T* Time
| *W* Katakana
| *X* Alphabetic only (character)
| *Y* Numeric only
| *Z* Timestamp

| **Date and time fields indicator**. Whether this format contains date and time fields. The possible values are:

| *0* The format does not contain date and time fields.
| *1* The format contains date and time fields.

| **Date and time format**. This value applies only to date, time, and timestamp fields. The possible values are:

| **USA* IBM USA standard (mm/dd/yyyy, hh:mm a.m., hh:mm p.m.)
| **ISO* International Standards Organization (yyyy-mm-dd, hh.mm.ss)
| **EUR* IBM European Standard (dd.mm.yyyy, hh.mm.ss)
| **JIS* Japanese Industrial Standard Christion Era (yyyy-mm-dd, hh:mm:ss)
| **SAA* SAA timestamp
| **MDY* Month/day/year (mm/dd/yy)
| **DMY* Day/month/year (dd/mm/yy)
| **YMD* Year/month/day (yy/mm/dd)
| **JUL* Julian (yy/ddd)
| **HMS* Hour/minute/second (hh:mm:ss)

| **Date and time separator**. This value applies only to date or time fields. The possible values are:

| */* Slash separator
| *-* Dash separator
| *.* Period separator
| *,* Comma Separator
| *:* Colon separator
| *(blank)* Blank separator

Decimal position. The number of decimal positions. This entry is zero if the field is not numeric.

| **Digits**. The number of 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

| **Graphic fields indicator**. Whether this format contains graphic fields. The possible values are:

| *0* The field does not contain graphic fields.
| *1* The field does contain graphic fields.

| **Input buffer position**. The field's position within the input record.

| **Internal field name**. The length of the alternative field name definition.

| **Null-capable fields indicator**. Whether this format contains null-capable fields. The possible values are:

| *0* The format does not contain null-capable fields.
| *1* The format contains null-capable fields.

| **Null values allowed**. Whether the result of this field can be the null value. The possible values are:

| *0* The field does not allow the null value.
| *1* The field does allow the null value.

| **Number of DBCS characters**. The number of DBCS characters this field can contain if the field type is graphic data type. This value does not include the 2 bytes for the variable length portion of the field.

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.

Reserved. An ignored field.

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.

Variable field indicator. Whether this field is variable length. Possible values are:

0 The field is not variable length.
1 The field is variable length.

Variable length fields in format indicator. Whether this format contains variable length fields. The possible values are:

0 The format does not contain variable length fields.
1 The format contains variable length fields.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C20 E Error found by program &1.
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.
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 returned. 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.
RCDL0300 Record format name and device file information. This format requires more system paging and takes longer to produce than the RCDL0100 format. This format is only applicable to device file types.

For more information, see "RCDL0100 List Data Section" on page 10-11, "RCDL0200 List Data Section" on page 10-11 or "RCDL0300 List Data Section" on page 10-11

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" on page 10-12.

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."

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(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 type
30	1E	CHAR(50)	File text description
80	50	BINARY(4)	File text description CCSID
84	54	CHAR (13)	File creation date

RCDL0100 List Data Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Format name

RCDL0200 List Data Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	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
82	52	CHAR(2)	Reserved
84	54	BINARY(4)	Record text description CCSID

RCDL0300 List Data Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Format name
10	A	CHAR(2)	Lowest response indicator
12	C	BINARY(4)	Buffer size
16	10	CHAR(20)	Record format type
36	24	CHAR(1)	Starting line number

Offset			
Dec	Hex	Type	Field
37	25	CHAR(1)	Separate indicator area present

Field Descriptions

Buffer size. The user buffer size.

Record text description CCSID.

0 There is no record text description.
1–65,535 The CCSID for the record text description.

File creation date. The date of the file in the format CYYMMDDHHMMSS, 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 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 text description CCSID.

0 There is no file text description.
1–65,535 The CCSID for the file text description.

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
RCDL0300 Record format name and device information

Lowest response indicator. The lowest response indicator in the file. The possible values are:

00 No response indicators in the file or response indicators are not applicable
01–99 Response indicator

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.

Record format type. The type of this record format. The possible values are:

Normal Normal record
SFL Subfile record
SFLMSGRCD Subfile message record
SFLCTL Subfile control record
USRDFN User-defined record
WINDOW Window record

Separate indicator area present. The existence of a separate indicator area. The possible values are:

0 No indicator area
1 Indicator area

Starting line number. A starting line number was specified for this record format. The possible values are:

0 Starting line number is not specified.
1 Starting line number is 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.
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.
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 returned. xx is from 01 to FF.
CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Process Extended Dynamic SQL (QSQPRCED) API

Parameters

Required Parameter Group:

1	SQL communications area	Output	Char(136)
2	SQL descriptor area	Input	Char(*)
3	Format name	Input	Char(8)
4	Function template	Input	Char(*)
5	Error code	I/O	Char(*)

The Process Extended Dynamic SQL (QSQPRCED) API provides functions to process extended dynamic SQL statements in an SQL package object.

Authorities and Locks

There are no authorities or locks for the Process Extended Dynamic SQL (QSQPRCED) API. Using an existing SQL package requires that you have *OBJOPR and *READ authority to the package. To use the PREPARE function of the API, you must have *OBJOPR and *ADD authority to the package.

Required Parameter Group

SQL communications area
OUTPUT; CHAR(136)
This is used for returning diagnostic information. It includes the SQLCODE variable, indicating whether an error has occurred. If SQLCODE has a value of 0 after a call to this API, the function was successful.

You should have this space declared in the program that calls this API. This parameter is considered output because the API uses the space to pass back information. The format of the structure is standard and can be included using the INCLUDE SQLCA statement in an SQL program. It is described more completely in the SQL/400 manuals.

SQL descriptor area
INPUT; CHAR(*)
This is used for you to pass information about the variables being used on a specific SQL statement. The SQLDA is used for passing the address, data type, length, and CCSID for variables on an OPEN, EXECUTE, FETCH, or DESCRIBE function.

The format of the structure is standard and can be included using the INCLUDE SQLDA statement in an SQL program. It is described more completely in the SQL/400 manuals.

Function template format
INPUT; CHAR(8)
The format of the function template being used. The possible value is:

SQLP0100 Basic template

For more information, see "SQLP0100 Format."

Function template
INPUT; CHAR(*)
A structure that determines the function to perform, the requested statement to process, and the SQL package to be used. This also contains the text of the statement, which is required for the PREPARE function. For the format of this parameter, see "SQLP0100 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.

SQLP0100 Format

The following shows the format of the function template parameter for the SQLP0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 10-14.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Function
1	1	CHAR(10)	SQL package name
11	B	CHAR(10)	Library name
21	15	CHAR(10)	Main program name
31	1F	CHAR(10)	Main program library name
41	29	CHAR(18)	Statement name
59	3B	CHAR(18)	Cursor name
77	4D	CHAR(1)	Open options
78	4E	CHAR(1)	Using clause for describe
79	4F	CHAR(1)	Commitment control

Offset			
Dec	Hex	Type	Field
80	50	CHAR(3)	Date format
83	53	CHAR(1)	Date separator
84	54	CHAR(3)	Time format
87	57	CHAR(1)	Time separator
88	58	CHAR(3)	Naming option
91	5B	CHAR(1)	Decimal point
92	5C	BINARY(2)	Blocking factor
94	5E	BINARY(2)	Statement length
96	5F	CHAR(*)	Statement text

Field Descriptions

Blocking factor. The number of records to be passed on a blocked FETCH request. The same number should be used on the OPEN and the FETCH request. The blocking factor is required for functions 4 and 5. It is ignored for other functions.

Commitment control. The commit level to be used. The possible values are:

C *CHG
S *CS
A *ALL
N *NONE

The commitment control value is required for function 1. It is ignored for other functions.

Cursor name. The name of the SQL cursor. The cursor name is required for functions 4, 5, 6, and 8. It is ignored for other functions.

Date format. The format used when accessing date result columns. All output date fields are returned in the format you specify. For input date strings, the value you specify is used to determine whether the date is a valid format. The valid values are:

USA IBM USA standard (mm.dd.yyyy, hh:mm a.m., hh:mm p.m.)
ISO International Standards Organization (yyyy-mm-dd, hh.mm.ss)
EUR IBM European Standard (dd.mm.yyyy, hh.mm.ss)
JIS Japanese Industrial standard Christion Era (yyyy-mm-dd, hh:mm:ss)
MDY Month/day/year (mm/dd/yy)
DMY Day/month/year (dd/mm/yy)
YMD Year/month/day (yy/mm/dd)
JUL Julian (yy/ddd)

The date format is required for function 1. It is ignored for other functions.

Date separator. The separator used when accessing date result columns. The valid values are:

/ Slash separator
. Period separator
, Comma separator
- Dash separator
blank Blank separator

The date separator is required for function 1. It is ignored for other functions.

Decimal point. The decimal point for numeric constants in SQL statements. The valid values are:

. Period separator
, Comma separator

The decimal point is required for function 1. It is ignored for other functions.

Function. The function being requested. The possible values are 1 through 9:

1 Build a new package into the specified library.
2 Prepare a statement into the specified package.
3 Execute a statement from the specified package.
4 Open a cursor defined by a prepared statement in a package.
5 Fetch data from an open cursor.
6 Close an open cursor.
7 Describe a prepared statement in a package.
8 Close an open cursor and delete the open data path
9 Prepare and describe in one step.

Library name. The library of the package.

Main program library name. The library of the main program.

Main program name. The name of the program representing the top program in the SQL application. When this program completes, all cursors are closed and the SQL environment goes away. This program must be on the stack or an error will occur (SQL0901). The main program name is required for all functions except 1. This allows you to control the boundary of the application.

Naming option. The naming convention used for naming objects in SQL statements. The valid values are:

SYS *library/file* syntax
SQL *collection/table* syntax

The naming option is required for function 1. It is ignored for other functions.

Open options. The open options used on an SQL cursor. These are specified using the following bits:

Bit(0) Read
Bit(1) Write
Bit(2) Update
Bit(3) Delete

For example, if a cursor is only for FETCH statements, the bit pattern should be '10000000'B or hex 80. If update capability is needed, the bit pattern should be '10100000'B. The syntax in the SQL statement takes precedence over the open options. This means that the FOR UPDATE OF and FOR FETCH ONLY clauses will be honored, even if they do not coincide with the requested open options. The open options are required for functions 2 and 4. They are ignored for other functions.

SQL package name. The name of the SQL package used as the repository for the extended dynamic SQL statements. The SQL package must not be a distributed SQL package created through the Create SQL Package (CRTSQLPKG) or the Create SQL xxx (CRTSQLxxx) commands. Attempted use of a distributed SQL package results in SQL0827. The SQL package name is required for all functions.

Statement length. The length of the SQL statement text that follows. The statement length is required for option 2. It is ignored for other functions.

Statement name. The name of the prepared SQL statement. The statement name is required for functions 2, 3, 4, 7 and 9. It is ignored for other functions.

Statement text. The SQL statement text that will be prepared. The statement text is required for option 2. It is ignored for other functions.

Time format. The format used when accessing time result columns. All output time fields are returned in the format you specify. For input time strings, the value you specify is used to determine whether the time is a valid format. The valid values are:

HMS	Hour/minute/second (hh:mm:ss)
USA	IBM USA standard (mm.dd.yyyy, hh:mm a.m., hh:mm p.m.)
ISO	International Standards Organization (yyyy-mm-dd, hh.mm.ss)
EUR	IBM European Standard (dd.mm.yyyy, hh.mm.ss)
JIS	Japanese Industrial standard Christian Era (yyyy-mm-dd, hh:mm:ss)

The time format is required for function 1. It is ignored for other functions.

Time separator. The separator used when accessing time result columns. The valid values are:

:	Colon separator
.	Period separator
,	Comma separator
blank	Blank separator

The time separator is required for function 1. It is ignored for other functions.

Using clause for describe. The value to assign to each SQLNAME variable in the SQLDA. The possible values are:

N	Column names
L	Column labels
B	Both (SQLDA must be allocated for twice as many entries)
A	Any labels that exist

These are explained more completely in the *SQL/400* Reference*. The using clause is required for functions 7 and 9. It is ignored for other functions.

Several options exist on an SQL precompile that cannot be specified on the creation of a package (function 1). For these options, the following defaults are used:

DLYPRP(*NO)
ALWCPYDTA(*YES)
ALWBLK(*READ).

Refer to the SQL/400 documentation for a full description of all the options.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C21 E Format name &1 is not valid.
SQL7023 E Parameter value not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 10-18 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 10-1 on page 10-17 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.

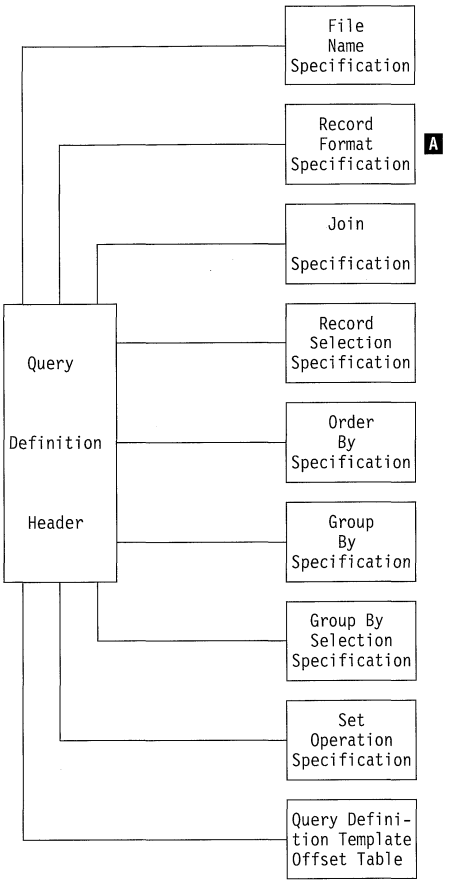


Figure 10-1. QDBQDT Format

Notice the box marked with an **A** in Figure 10-1. The topic "Format Definition (QDBFMTD)" 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 10-2 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 OPDBAPI, in the QUSRTOOL library.

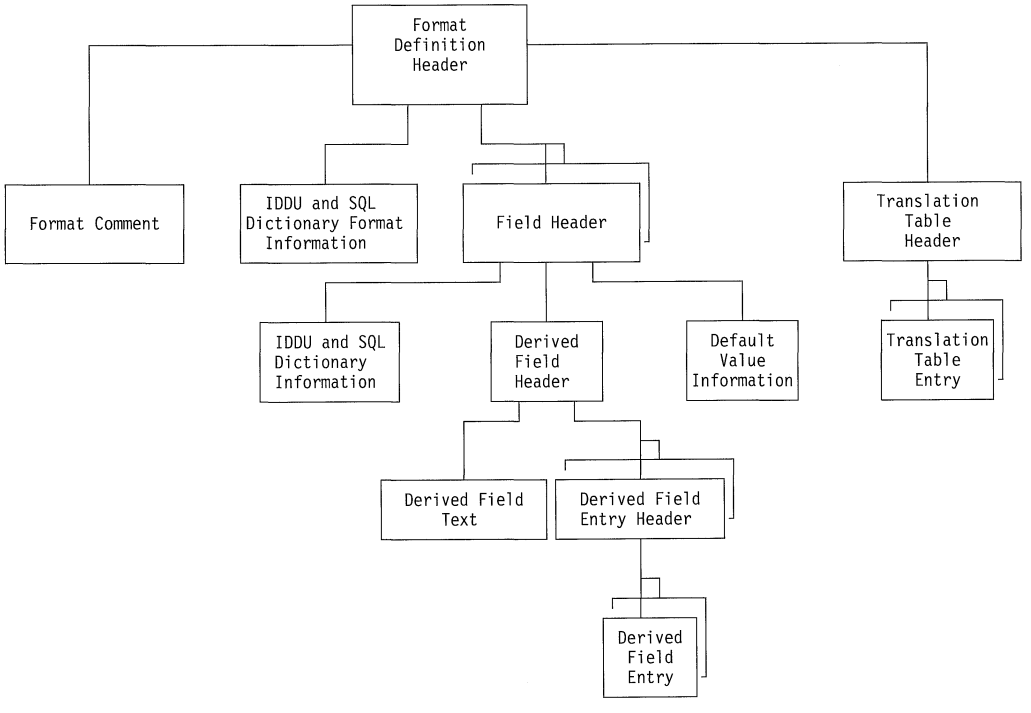


Figure 10-2. QDBFMTD Format

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 or activation group number
- AS/400 system environment
- Null-capable fields
- File dependency
- Level check
- Record format specifications
- Secure
- Shared
- Open scope

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

CPF2114 E Cannot allocate object &1 in &2 type *&3.
CPF2115 E Object &1 in &2 type *&3 damaged.
CPF2169 E Job's sort sequence information not available.
CPF24B4 E Severe error while addressing parameter list.
CPF2619 E Table &1 not found.
CPF3BCC E Language identifier &1 not valid.
CPF3BC6 E Sort sequence &1 not valid.
CPF3BC7 E CCSID &1 outside of valid range.
CPF3BC8 E Conversion from CCSID &1 to CCSID &2 is not supported.
CPF3BC9 E Conversion from CCSID &1 to CCSID &2 is not defined.
CPF3CF1 E Error code parameter not valid.
CPF3FC0 E Language identifier is not valid.
CPF4000 E All CPF40xx messages could be signalled. xx is from 01 to FF.
CPF4100 E All CPF41xx messages could be returned. xx is from 01 to FF.
CPF4200 E All CPF42xx messages could be returned. xx is from 01 to FF.
CPF4300 E All CPF43xx messages could be returned. xx is from 01 to FF.
CPF5000 E All CPF50xx messages could be signalled. xx is from 01 to FF.
CPF5100 E All CPF51xx messages could be returned. xx is from 01 to FF.
CPF5200 E All CPF52xx messages could be returned. xx is from 01 to FF.
CPF5300 E All CPF53xx messages could be returned. xx is from 01 to FF.
CPF8133 E Table &4 in &9 damaged.
CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.

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 the following format names:

FILD0100 File description template
FILD0200 Format definition
FILD0300 Key field information

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.
***FILETYPE** 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 the following formats:

FILD0100 File definition template
FILD0200 Format definition
FILD0300 Record format

The following sections provide an overview of each of these formats.

FILD0100 Format (File Definition Template (FDT) header): FILD0100 provides detailed information about how the file is built. Figure 10-3 on page 10-20 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 OPDBAPI, in the QUSRTOOL library.

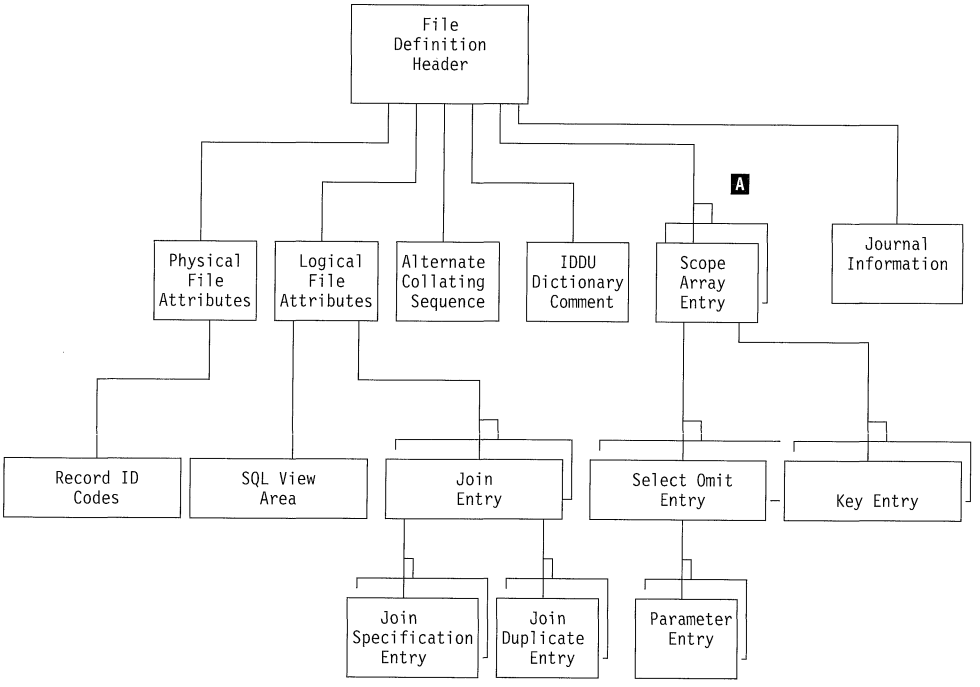


Figure 10-3. FILD0100 Format

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 10-4 on page 10-21 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 OPDBAPI, in the QUSRTOOL library.

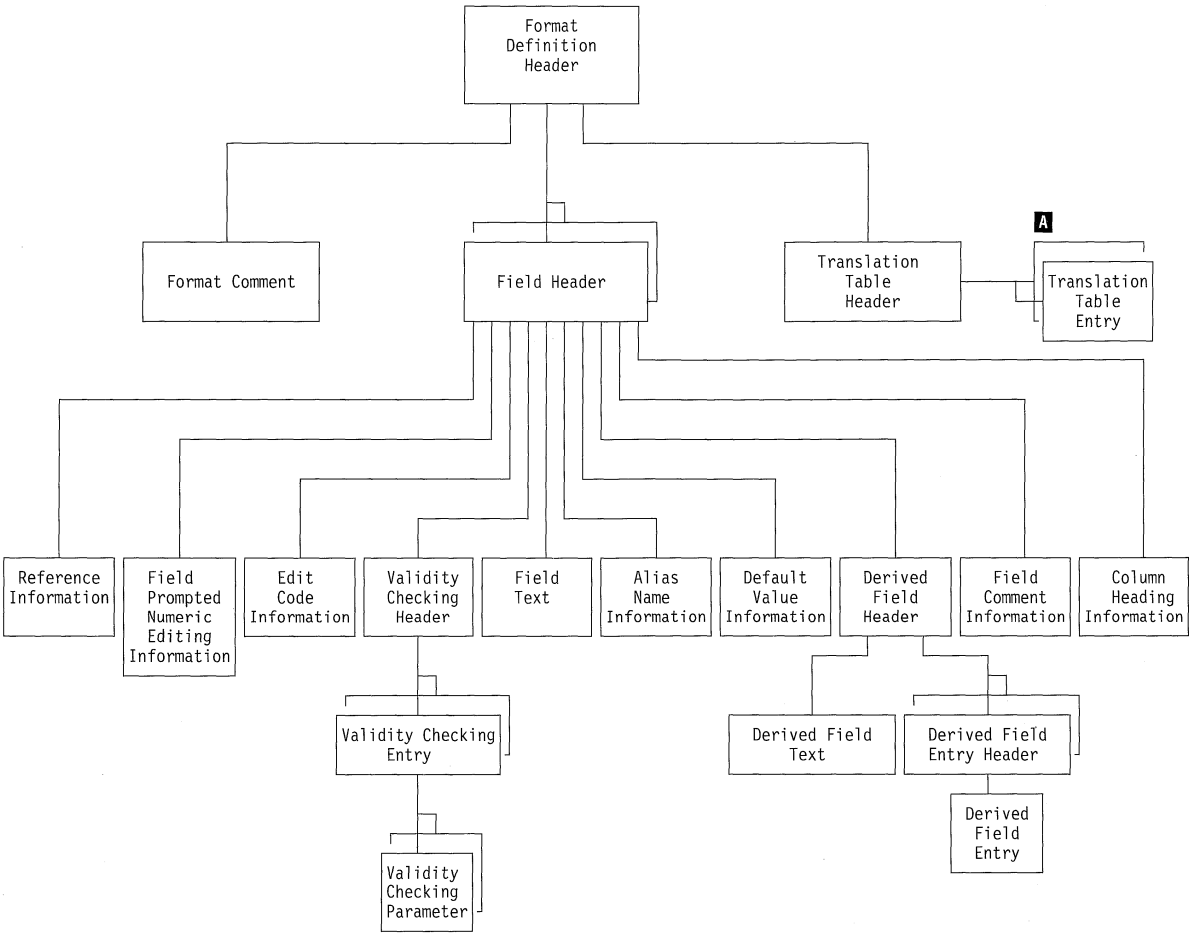


Figure 10-4. FILD0200 Format

FILD0300 Format (Key Field Information): FILD0300 provides detailed information for key fields of each record format of the specified file. This structure is used by the QQQQRY API to get data from the named file. Figure 10-5 on page 10-22 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 OPDBAPI, in the QUSRTOOL library. Note that an offset to the key field information array of each record format is provided in the record format information structure. If 0 is returned for this offset, this record format has no key field. If -1 is returned for this offset, the size of the receiver provided is insufficient to hold the return data.

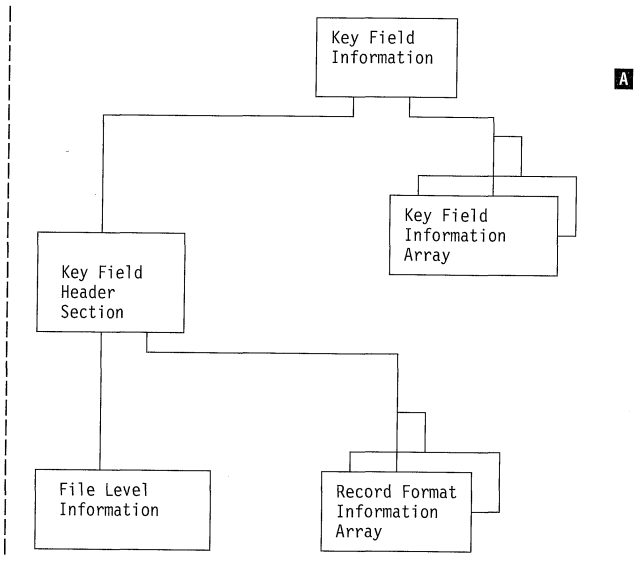


Figure 10-5. FILD0300 Format

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.
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.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Member Description (QUSRMBRD) 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	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" or "MBRD0300 Format" on page 10-24.

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 10-24.

MBRD0100 Format: The MBRD0100 format includes the file member list and source information shown in the following table.

Offset		Type	Field
Dec	Hex		
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
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

Offset			
Dec	Hex	Type	Field
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	BINARY(4)	Member text description CCSID
244	F4	CHAR(22)	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			
Dec	Hex	Type	Field
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			
Dec	Hex	Type	Field
		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
		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.

Member text description CCSID. The CCSID for the member text description. The job default CCSID of the current process will be used to translate the text. For more information about CCSID, see the *National Language Support Planning Guide*.

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 10-24 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

| 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.
| 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.
| CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
| CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.

Chapter 11. Commitment Control APIs

Commitment control allows you to define and process changes to resources, such as database files or tables, as a single transaction (logical unit of work).

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 to retrieve information about commitment control.

Commit or rollback operations include all methods of commit and rollback, such as:

- CL COMMIT and ROLLBACK commands
- C/400_Rcommit and _Rrollback functions
- SQL/400 COMMIT and ROLLBACK statements

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, that resource can be referred to as an **API commitment resource**.

API commitment resources are processed by the system during commit, rollback, process end, activation group end, and, optionally, during an initial program load (IPL).

When commitment control is started using the Start Commitment Control (STRCMTCTL) command, the system creates a **commitment definition** that is scoped to a particular activation group or to the job as indicated on the commit scope (CMTSCOPE) keyword.

Each group of committable changes is intended to be an atomic operation. Each group can be committed (changes are made permanent to the system) or rolled back (changes are permanently removed from the system) and is referred to as a transaction (logical unit of work). The first transaction begins when commitment control is started. Each commit and rollback completes the current transaction and starts a new transaction.

A commitment definition saves internal control information pertaining to the resources under commitment control. This internal control information is maintained as the state of those commitment resources changes, until that commitment definition is ended using the End Commitment Control (ENDCMTCTL) command. A commitment definition generally includes:

- The parameters on the Start Commitment Control (STRCMTCTL) command
- The current status of the commitment definition
- Information about files and other resources that contain changes made during the current transaction

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 APIs include the following:

Add Commitment Resource (QTNADDCR) adds an API commitment resource to a commitment definition.

Remove Commitment Resource (QTNRMVCR) removes an API commitment resource from a commitment definition.

Retrieve Commitment Information (QTNRCMTI) gets status and lock-level information about the currently active commitment definition for the program making the retrieve request.

Figure 11-1 on page 11-3 shows how the commitment control APIs can be used together. First, the Retrieve Commitment Information (QTNRCMTI) API is used by the high-level language (HLL) program to determine if commitment control is active within the activation group for the HLL program. If the activation-group-level commitment definition is already active, then the status retrieved by the API will be with respect to that activation-group-level commitment definition. If the activation-group-level commitment definition is not active, but the job-level commitment definition is active, then the status retrieved by the API will be with respect to the job-level commitment definition. If status is being retrieved for the job-level commitment definition, then information from a second status field returned by the API can be used to determine whether programs that have run within the activation group have already used the job-level commitment definition. If no program running within the activation group has used the job-level commitment definition, then an activation-group-level commitment definition may be started by the HLL program.

Note: Programs running within a single activation group may use the activation-group-level or the job-level commitment definition, but cannot use both definitions concurrently. Two programs running within different activation groups may each use a separate activation group level commitment definition, or one or both programs may use the job-level commitment definition.

Once a commitment definition has been established for the HLL program, the Add Commitment Resource (QTNADDCR) API is used to add one or more commitment resources to the commitment definition. When a commit or rollback operation is performed to complete a transaction for this commitment definition, the system performs the commit or rollback operation for all record-level and object-level resources. It also calls an exit program, as identified by the HLL program when the API commitment resource was added, for each API commitment resource.

After all the desired transactions are completed by the HLL program, the Remove Commitment Resource (QTNRMVCR)

API must be used to remove each of the commitment resources added to the commitment definition before the commitment definition can be ended by the End Commitment Control (ENDCMTCTL) command. However, if an activation-group-level commitment definition is being used and the activation group is ended when the HLL program returns, then any API commitment resources are implicitly removed by the system and the activation-group-level commitment definition is automatically ended by the system. Prior to the system implicitly removing the API commitment resources and automatically ending the activation-group-level commitment definition, an implicit commit or rollback operation is performed by the system if pending changes exist for the commitment definition, with the appropriate exit program calls made for any API commitment resources. An implicit commit is performed by the system if the activation group is ending normally. An implicit rollback is performed by the system if the activation group is ending abnormally.

Note: The implicit end performed by the system for activation-group-level commitment definitions does not apply for the default activation group. This is because the default activation group is never ended when a program running within it returns. The default activation group persists for the life of the job.

Regardless of the scope for a particular commitment definition, any pending changes for a commitment definition at process end or during IPL recovery processing are always rolled back. This is true unless the process or system ended in the middle of a commit operation for that commitment definition. In that case, the commit operation is completed for the commitment definition.

This chapter presents the APIs in alphabetical order, followed by a description of the Commit and Rollback exit program.

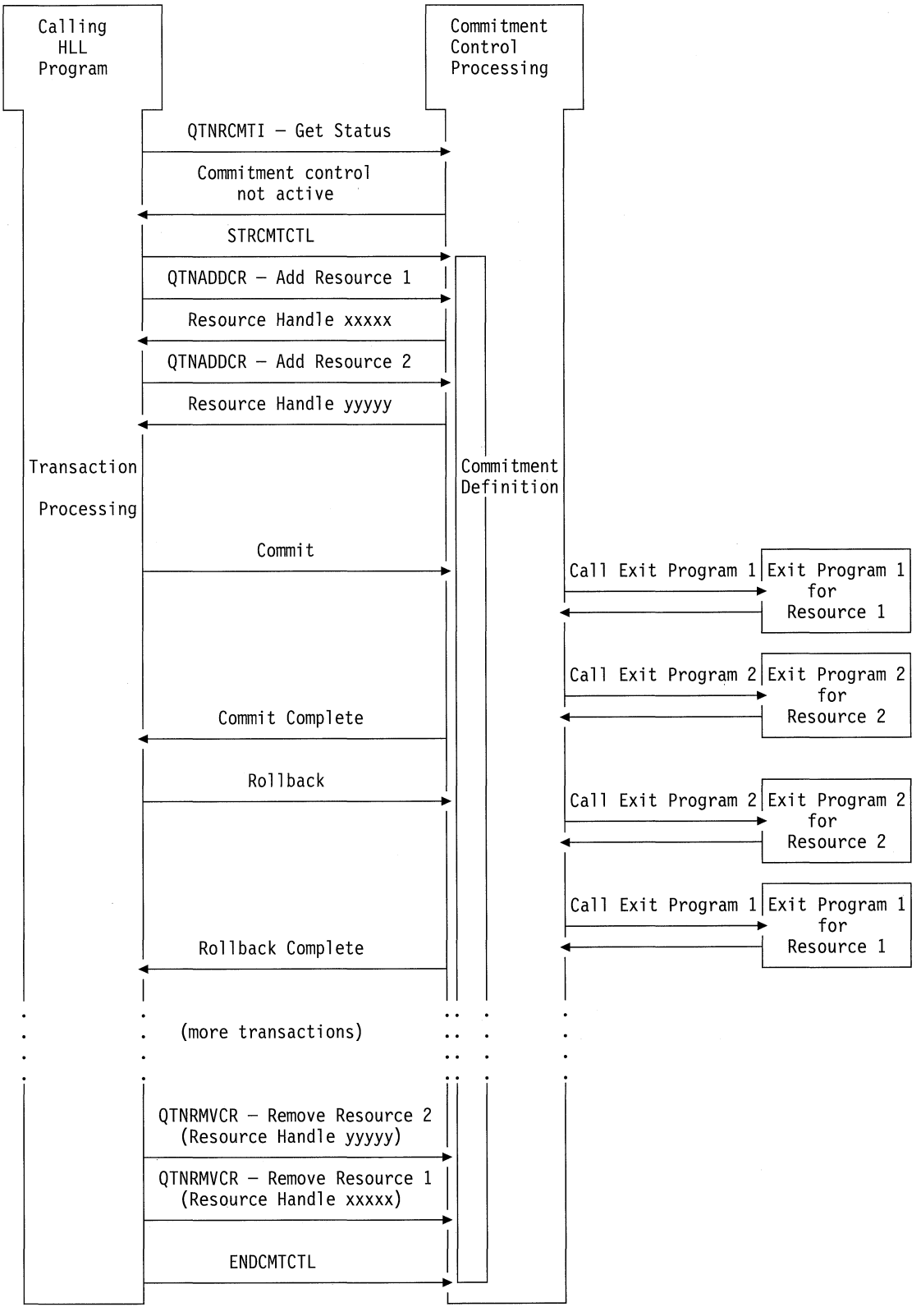


Figure 11-1. Example Using the Commitment Control APIs

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 a commitment definition. When the resource has been added, the specified exit program is called for each commit or rollback operation performed for the commitment definition until the resource is removed. Once an API commitment resource is added, it must be removed with the Remove Commitment Resource (QTNRMVCR) API before commitment control can be ended for the commitment definition, unless activation group level commitment definitions are used. Activation group level commitment definitions for non-default activation groups are automatically ended by the system and any API commitment resources implicitly removed when the activation group is ended. See "Remove Commitment Resource (QTNRMVCR) API" on page 11-5 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 to that particular commitment definition. 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.

For more information about the exit program and information that is passed to it, see "Commit and Rollback Exit Program" on page 11-8.

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 when 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 11-8.

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.

If the exit program is to be called during IPL processing, the information passed-in or pointed-to by this parameter must not be temporary. That is, the information referred to and used by the exit program must persist across an IPL.

IPL processing option
INPUT; CHAR(1)
Indicates whether the Commit and Rollback exit program

will be called during any commit or rollback processing that occurs during IPL recovery processing for the commitment definition.

N If the API resource is in the commitment definition when the system ends abnormally, the Commit and Rollback exit program is not called during the IPL recovery processing for the commitment definition.

Y If the API resource is in the commitment definition when the system ends abnormally, the Commit and Rollback exit program is called during the IPL recovery processing for the commitment definition.

Note: When called during IPL, the exit program runs under the same user profile that originally added the commitment resource.

When the Commit and Rollback exit programs are called during IPL recovery processing for a particular commitment definition, the exit programs are called in the order they were added to the commitment definition if a commit operation is to be completed. The Commit and Rollback exit programs are called in the reverse order they were added to the commitment definition if a rollback operation is to be performed. When multiple commitment definitions from a single process need recovery at IPL time or at process end, there is no guaranteed order of exit program calls across commitment definitions. The ascending or descending order of exit program calls is only guaranteed within each 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 adding a commitment resource using this API when:

- Commitment control is not active for the program when making the request to add a commitment resource.
- Distributed data management (DDM) remote resources are under commitment control for the same process and commitment definition.[1]
- Distributed relational database remote resources are under commitment control for the same process and commitment definition[1].
- Commitment control cannot get a shared-no-update (*SHRNUP) lock on the Commit and Rollback exit program.
- Commit or rollback processing is currently in progress for the commitment definition that is to have the commitment resource added.
- The checkpoint processing for a save-while-active function is in progress in another job.

In all other instances, the API commitment resource is added to the commitment definition.

Once a resource has been added to a commitment definition, the process must not change the authorities to the Commit and Rollback exit program or delete the exit program.

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.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
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 a commitment definition using the Add Commitment Resource (QTNADDCR) API. For more information about adding resources to a commitment definition, see "Add Commitment Resource (QTNADDCR) API" on page 11-4.

Once a commitment resource is removed, the resource handle that refers to it is no longer valid. You cannot end commitment control for a commitment definition until all API commitment resources have been removed.

If an End Job (ENDJOB) command is entered or you sign off the job, the system automatically ends commitment control for all commitment definitions for the job. Likewise, the system will automatically end an activation group level commitment definition for a non-default activation group that is

[1] You can use the Retrieve Commitment Information (QTNRCMTI) API to retrieve information about what type of commitment control resources are currently associated with the currently active commitment definition for the program making the retrieve request. See "Retrieve Commitment Information (QTNRCMTI) API" on page 11-6 for more information.

ending. Any API commitment resources that have not yet been removed from any commitment definition being automatically ended by the system will be implicitly removed by the system during the end job or the activation group end processing. Prior to the system implicitly removing the API commitment resources and automatically ending a commitment definition, an implicit commit or rollback operation is performed by the system if pending changes exist for the commitment definition, with the appropriate exit program calls made for any API commitment resources. An implicit commit is performed by the system if the activation group is ending normally. An implicit rollback is performed by the system if the activation group is ending abnormally or the job is ending. For more information about the exit program and information that is passed to it, see "Commit and Rollback Exit Program" on page 11-8.

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 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 program making the request to remove the commitment resource.
- Commit or rollback processing is currently in progress for the commitment definition that is to have the commitment resource removed from the commitment definition.

In all other instances, the API commitment resource is removed from the commitment definition.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF8362 E Request for commit resource is not valid; reason code &1.
CPF8367 E Cannot perform commitment control operation.
CPF9872 E Program &1 in library &2 ended. Reason code &3

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/0	Char(*)

The Retrieve Commitment Information (QTNRCMTI) API allows you to determine if commitment control is active within the activation group for the program performing the retrieve request. In addition, the default lock level and scope for the commitment definition can also be retrieved.

Information about commitment definitions that are started by the system for system use only 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" on page 11-7.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned

Offset		Type	Field
Dec	Hex		
4	4	BINARY(4)	Bytes available
8	8	CHAR(1)	Commitment definition status
9	9	CHAR(10)	Commitment definition default lock level
19	13	CHAR(1)	Commitment definition scope
20	14	CHAR(1)	Commitment definition status for the activation group

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 at the start of commitment control for the commitment definition. This is the level of record locking that applies to records in all commitment resources under commitment control for the commitment definition. The level of record locking is this value unless it is overridden when the file is opened. You cannot override this value; however, the system can override for files opened for system functions. The possible values are:

blank Blanks are returned when I is returned for commitment definition status.
**ALL* Changed and retrieved records are protected from changes by other jobs running at the same time.
**CHG* Changed records are protected from changes by other jobs running at the same time.
**CS* Changed and retrieved records are protected from changes by other jobs running at the same time. Retrieved records are protected until they are released or until a different record is retrieved.

Commitment definition scope. The scope for the commitment definition currently active within the activation group for the program performing the retrieve request. The possible values are:

A Activation group level
J Job level

Commitment definition status. The overall status of the commitment definition currently active for the activation group for the program performing the retrieve request. The scope for this commitment definition is returned in the commitment definition scope field. The possible values are:

I Commitment control is not active at either the activation group level or the job level for the program making the retrieve request.
A The commitment definition is active within the activation group for the program performing the retrieve request. No local, remote, or API commitment resource is associated with the commitment definition.
L The commitment definition is active on the local system within the activation group for the program performing the retrieve request. An *L* is returned if one or more of the following resources are under commitment control.
- Local, open database files with pending changes.
- Local, closed database files with pending changes.
- Resources with object-level changes.
- Local distributed relational database resources.
- API commitment resources.

R The commitment definition is active on a remote system within the activation group for the program performing the retrieve request. An *R* is returned if one or more of the following resources are under commitment control.
- Remote DDM resources.
- Remote distributed relational database resources.

Commitment definition status for the activation group. The status of the commitment definition currently active for the activation group for the program performing the retrieve request, but specifically regarding any commitment resource changes made by programs running within the activation group. The possible values are:

I Commitment control is not active at the activation group level or the job level for the program making the retrieve request. The overall status for the commitment definition is also not active.
A The commitment definition is currently being used by one or more programs running within this activation group. Local, remote or API commitment resources may have been placed under commitment control to the commitment definition by a program running within the activation group. The overall status for the commitment definition is either active (A), or active on the local system (L), or active on a remote system (R).
N The job-level commitment definition is active but is not in use by any program running within this activation group. The overall status for the commitment definition is either active (A), or active on the local system (L), or active on a remote system (R). It is possible to start the activation-group-level commitment definition for programs running within this activation group.

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.
CPF3C24 E Length of the receiver variable is not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3

Commit and Rollback Exit Program

Parameters

Required Parameter Group:

1	Commit and Rollback exit program information	Input	Char(80)
2	Status information	Input	Char(*)

Users who add API commitment resources to the commitment definition must supply a Commit and Rollback exit program as described in the qualified Commit and Rollback exit program name parameter on page 11-4. 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."

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 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(*)
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 for IPL recovery
6	6	CHAR(4)	Reserved
10	A	CHAR(1)	Process error status
11	B	CHAR(1)	Process end status
12	C	CHAR(1)	Reserved
13	D	CHAR(1)	Commit or rollback qualifier
14	E	CHAR(1)	Commitment definition scope
15	F	CHAR(17)	Reserved

Field Descriptions

Called for IPL recovery. Indicates if the exit program was called to perform IPL recovery processing for the API commitment resource. The possible values are:

Y Called to perform IPL recovery processing for the API commitment resource
N Not called to perform IPL recovery processing for the API commitment resource

Commit or rollback. Indicates if 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

Commit or rollback qualifier. Indicates if the commit or rollback operation is being performed on behalf of an explicit request by a program or is being performed implicitly by the system.

E Explicit commit or rollback
I Implicit commit or rollback

Commitment definition scope. The scope for the commitment definition. The possible values are:

A Activation group level
J Job level

Process end status. Indicates if the exit program was called because of process end, and if so, how the process is ending, or if the exit program was called as the result of an activation group ending. The possible values are:

0 Not during the process or activation group end
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
4 Activation group is ending

Process error status. Indicates if 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

Reserved. An ignored field.

Status information length. The length in bytes of all data passed to the Commit and Rollback exit program.

Exit Program Locks

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. 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 process end or IPL recovery processing, the system does not allow a Commit and Rollback exit program to run more than 5 minutes. An exit program will not be allowed to prevent a process from ending or an IPL from completing.
- 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 an escape message to commitment control, the system assumes there is a failure. A diagnostic message is returned to the calling program or sent to the QHST log if at IPL time.
- Perform any necessary cleanup of locks acquired by the exit program. This is important if the exit program is called during IPL processing.
- Be written expecting to be called as part of every commit and rollback that is performed for a commitment definition, including implicit commit and rollback operations performed by the system at:
  - Activation group end
  - Job end
  - IPL time (optionally)

Your exit program **must not** perform any of these operations if the scope for the commitment definition is the job level, or any of these functions from the same activation group if the scope for the commitment definition is the activation group level.

- 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 of these functions if the scope for the commitment definition is the job level, or any of these functions from the same activation group if the scope for the commitment definition is the activation group level.

- 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). Perform any necessary cleanup of locks acquired by the exit program.
- Prevent the potential for any infinite looping conditions. The system stops the exit program, after 5 minutes, during process end or IPL time.
- 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 future calls to the exit program.

Process End, Activation Group End, and IPL Recovery Processing Guidelines: During process end, activation group end, and IPL recovery processing, the debug functions are not available to help debug any exit program problems. The following operations may be performed during these processing phases. 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, including creating, changing, opening, closing, clearing, and deleting.
- Database input and output operations.
- 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, activation group end, or IPL are:

- Signal any inquiry messages.
- Submit any other jobs.
- Use or attempt to start any remote communications activities.

- Start any subsystems.

Part 5. Debugger APIs

Chapter 12. Debugger APIs

The debugger APIs can be used to write debuggers for the AS/400 system. The users of these APIs include:

- The source debugger that is shipped with the OS/400 licensed program
  - A **source debugger** is a tool for debugging Integrated Language Environment* (ILE*) programs by displaying a representation of their source code.
- Any other debugger that IBM or a business partner writes

Debugger functions are designed to help you write and maintain your applications. All APIs and exit programs in this chapter are designed to debug ILE applications unless otherwise noted. You can run your programs in a special testing environment while closely observing and controlling the processing of these programs in the testing environment. You can write a debugger application that interacts with the APIs provided in this chapter, or you can use the debugger provided with the AS/400 system.

No special commands specifically for testing are contained in the program being tested. The same program being tested can run normally without changes. All debugger APIs are specified within the job the program is in, not as a permanent part of the program being tested. With the debugger APIs provided, you interact with your programs symbolically in the same terms as the high-level language (HLL) in the program. You refer to variables by their names and to locations as the line and the column within a view. In addition, the debugger functions are only applicable to the job in which they are set up. The same program can be used at the same time in another job without being affected by the debugger functions set up.

To use a debug API, the application must bind to the service program QTEDBGS in QUSRTOOL. All API functions listed in this chapter are then available to the application.

How a source debugger application uses the APIs to debug ILE programs: The Start Debug command has a parameter, SRCDBGPGM, that specifies which program is called when an ILE program is debugged. The system calls this program, indicating that the debug session is to begin. It also calls this program when the user wants to show the Display Module Source display.

When the system calls the source debugger program, indicating the start of a debug session, that program uses source debugger APIs to perform debug functions. The first API that is called is the Start Source Debug (QteStartSourceDebug) API, which indicates to the system that a source debugger is running.

When an ILE program is debugged, the Retrieve Module Views (QteRetrieveModuleViews) API is used to obtain information about the views available in the modules of that program. A **view** is text that is displayed by the source debugger. A module may have several views, depending on the debug data supplied by the compiler of that module. See the appropriate language reference manual to determine which views are available.

Every module has at least one view, the statement view. A **statement view** is a low-level view that contains information about each high-level statement in that module. This view is not meant to be displayed, although there is text associated with that view. The information in the statement view text can be used by the source debugger to determine the following:

- Procedure name
- Statement number
- Statement type associated with any high-level language statement in the module

The source debugger application uses the Register Debug View (QteRegisterDebugView) API to register the views of a program. Once these views are registered, various debug operations can be performed against these views. These operations include:

- Retrieving the text associated with the views
- Adding a breakpoint to the program at a certain location in a view
- Displaying variables that are defined in the program

The source debugger application uses the Retrieve View Text (QteRetrieveViewText) API to retrieve the text of a view. Every view has text associated with it that can be retrieved using the QteRetrieveViewText API.

When a program is being debugged and it stops at a breakpoint, the system indicates that it has stopped by calling the Program-Stop Handler exit program. This program is passed a line number in the statement view where the program being debugged has stopped.

The Map View Position (QteMapViewPosition) API is used to map positions in one view to positions in another view. For example, if the source debugger is currently displaying a source view in a module, and a breakpoint occurs, the Program-Stop Handler exit program is called. This program is passed a line number in the statement view of that module, which indicates at which statement the program has stopped. To show the position in the source view where the program has stopped, the application maps the statement view position to a source view position.

When the debug session is over, the source debugger application issues the End Source Debug (QteEndSourceDebug) API, which removes all ILE programs from debug mode. No source debugger APIs can be issued until the source debug session is started again with the Start Debug command.

The APIs are divided into the following functional areas:

- **Debug Session Control APIs and Exit Programs**

  These APIs are used to:

  - Start and end the source debug session
  - Determine which programs, modules, and views are referenced
  - Control certain attributes of the debug environment

  Start Source Debug (QteStartSourceDebug) enables your session to use the source debugger.
  End Source Debug (QteEndSourceDebug) takes the job out of debug mode.
  Retrieve Debug Attribute (QteRetrieveDebugAttribute) retrieves the attributes of the source debug session.
  Retrieve Module Views (QteRetrieveModuleViews) returns to the caller the list of modules and views associated with a specific ILE program.
  Program Stop Handler exit program is registered on the Start Source Debug API. This program is called by the source debugger support when an ILE program stops at a breakpoint or for other reasons.
  Debug Session Handler exit program manages the ILE debugger, telling it when to start, stop, and display its screens. This program is registered on the Start Debug Command.

- **View Information APIs**

  These APIs retrieve view information, including view text information and view mapping information, and allow the program to set parameters associated with a view.

  Register Debug View (QteRegisterDebugView) registers a view of a module, which allows a program to be debugged in terms of that view.
  Remove Debug View (QteRemoveDebugView) removes a view of a module that was previously registered by the Register Debug View API. This is necessary when a program is to be removed from debug mode so it can be deleted and recompiled.
  Map View Position (QteMapViewPosition) maps positions from one view to another view within the same program and module.
  Retrieve View Text (QteRetrieveViewText) retrieves source text from the specified view.
  Retrieve Stopped Position (QteRetrieveStoppedPosition) determines if a program is on the call stack and indicates the position in the view at which the program is stopped if it is on the stack.

- **Debug Command API**

  Debug commands are a part of the API that takes on free-form expressions. They are referred to as the debug language that the program may supply to the source debugger support.

  Submit Debug Command (QteSubmitDebugCommand) allows a program to issue debug language statements. Debug language statements permit programs to enter breakpoints, run one or more statements of a program being debugged, and evaluate expressions.

- **Original Program Model (OPM) API**

  Retrieve Program Variable (QTERTVPV) retrieves the current value of one program variable in a program that is being debugged. This API supports OPM programs only. It cannot be used if the user is servicing another job and that job is on a job queue, or is held, suspended, or ended.

For general information about the Integrated Language Environment, see *ILE* Concepts*.

The API descriptions are presented in alphabetical order.

End Source Debug (QteEndSourceDebug) API

Parameter

Required Parameter:

1	Error code	I/O	Char(*)

The End Source Debug (QteEndSourceDebug) API is used to end the source debug support. All ILE programs being debugged are removed from debug mode. All registered views to programs being debugged are no longer valid.

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

CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Internal error(s) occurred.
CPF9541 E Not in debug mode.
CPF9549 E Error addressing API parameter.

Map View Position (QteMapViewPosition) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	From view ID	Input	Binary(4)
4	From line number	Input	Binary(4)
5	From column number	Input	Binary(4)
6	To view ID	Input	Binary(4)
7	Error code	I/O	Char(*)

The Map View Position (QteMapViewPosition) API maps positions from one view to another view within the same program and module. A specified position in the view identified in the from view ID parameter is used for the mapping. The position is specified as a line number and a column number in the from view ID.

A position in one view can map to more than one position in another view. For example, an SQL statement in the SQL input source view may map to many positions in the C input source view. This is because a single SQL statement may distribute source throughout the output of the SQL processor.

One or more positions in the to view ID are returned as a line number and a column number.

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. For more information on the size and format of the receiver variable, see "Format of Receiver Variable."

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.

It is suggested that a receiver variable length be given that is large enough to hold one map element. Because this is normally the number of elements returned, a single call to this API is usually sufficient.

From view ID
INPUT; BINARY(4)
The identifier of a previously registered view, which is obtained using the Register Debug View API. This ID specifies the from view in the mapping function provided.

From line number
INPUT; BINARY(4)
The line number in the view specified by the from view ID parameter mapped to a line number in the view specified by the to view ID parameter.

If the information in the from view ID parameter is a statement view, this parameter represents the line number in the statement view.

Note: The statement view is the lowest level view. Breakpoints, steps, and unmonitored exceptions are reported as a line number within this view. Therefore, the statement view must exist and be registered to accomplish source level debugging.

From column number
INPUT; BINARY(4)
The position in the line specified by the from line number parameter. Column numbers cannot exceed 255.

If the from view ID parameter is a statement view, this parameter is not used and should be set to column one.

To view ID
INPUT; BINARY(4)
The identifier of a previously registered view, which is obtained using the Register Debug View API. This specifies the to view in the mapping function provided.

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 Receiver Variable: The following table shows the information supplied in the receiver variable parameter.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of map elements
*	*	BINARY(4)	Line number
*	*	BINARY(4)	Column number

Field Descriptions

Bytes available. The number of bytes of data available to be returned to the user.

Bytes returned. The number of bytes of data returned to the user.

Column number. The column number within the from line number parameter that maps to the current position in the to view ID parameter. Column numbers are 1 through 255.

If the view is a statement view, this number is not used and is set to column one.

Line number. The line number in the view specified by the to view ID parameter.

Number of map elements. The line number and column number fields and are repeated this number of times, once for each map available.

Error Messages

CPF3C24 E Receiver variable length not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Internal error(s) occurred.

| CPF9541 E Not in debug mode.
| CPF9543 E From view not found.
| CPF9544 E To view not found.
| CPF9548 E Map not available.
| CPF9567 E Column number not valid.
| CPF9568 E Line number not valid.
| CPF9549 E Error addressing API parameter.

Register Debug View (QteRegisterDebugView) API

Parameters

Required Parameter Group:

1	View ID	Output	Binary(4)
2	Number of lines	Output	Binary(4)
3	Returned library	Output	Char(10)
4	View timestamp	Output	Char(13)
5	Program name	Input	Char(20)
6	Program type	Input	Char(10)
7	Module name	Input	Char(10)
8	View number	Input	Binary(4)
9	Error code	I/O	Char(*)

The Register Debug View (QteRegisterDebugView) API registers a view of a module, which allows a program to be debugged in terms of that view. An identifier to the view is returned on successful completion of the API to be used in subsequent view information APIs.

If a request is made to register an already registered view, no error occurs. Instead, the previous ID is returned.

Note: Before registering views for a program again, it is recommended that all views for that program first be removed.

Required Parameter Group

View ID
OUTPUT; BINARY(4)
The returned ID of the successfully (or previously) registered debug view. The source debugger support supplies and maintains the view IDs. If no error is reported by the API, this value is used by the program in view ID input parameters that occur on subsequent debugger APIs.

Number of lines
OUTPUT; BINARY(4)
The returned number of lines of text in the view.

Returned library
OUTPUT; CHAR(10)
The library where the program was found. This is useful when *LIBL or *CURLIB is specified for the program library.

View timestamp
OUTPUT; CHAR(13)
The date and time the view was created. If this time is greater than the time obtained from the Retrieve Module Views API, the view may not be the same as the previous one. Users should run the Retrieve Module Views API before registering the view. The value is the American National Standard 13-character timestamp.

Program name
INPUT; CHAR(20)
The name of a program for which a view is to be registered. The first 10 characters contain the program name. The second 10 characters contain the name of the library where the program is located. The following special values may be used for the library name:

***CURLIB** The job's current library.
***LIBL** The library list.

Program type
INPUT; CHAR(10)
The type of program for which a view is to be registered. This is the object type of the program object. The valid values are:

***PGM** Callable program
***SRVPGM** Service program

Module name
INPUT; CHAR(10)
The name of a module for which a view is to be registered. The module name should be left-justified.

Information for this parameter is available by using the Retrieve Module Views API to retrieve available module names for a specified program.

View number
INPUT; BINARY(4)
The number of a view to be registered for subsequent view information and debug command APIs. If -1 is specified, the statement view is registered. The value -1 is a shortcut to allow the registering of this view without going through the Retrieve Module Views API to obtain the number.

Information for this parameter is available by using the Retrieve Module Views API to retrieve available view numbers for modules associated with a specific program.

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.
| CPF3CF2 E Internal error(s) occurred.
| CPF9541 E Not in debug mode.
| CPF9542 E View not found.
| CPF954F E Module not found.

| CPF9562 E Cannot debug module.
| CPF955F E Not an ILE program.
| CPF9801 E Object not found.
| CPF9802 E Not authorized to use object.
| CPF9809 E Library cannot be accessed.
| CPF9810 E Library not found.
| CPF9820 E Not authorized to use library.
| CPF9549 E Error addressing API parameter.

Remove Debug View (QteRemoveDebugView) API

Parameters

Required Parameter Group:

1	View ID	Input	Binary(4)
2	Error code	I/O	Char(*)

The Remove Debug View (QteRemoveDebugView) API removes a view of a module that was previously registered by the Register Debug View API. This API is needed when a program is to be removed from debug, so that it can be deleted and recompiled. Once a view is removed from being debugged, its view number may not be used again.

If the last registered view of a program is removed, all breakpoints are removed from that program, and the step statement is disabled if it was active.

Required Parameter Group

View ID
INPUT; BINARY(4)
The ID of a view to be removed from debug. This ID was obtained from a previous use of the Register Debug View 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.
CPF3CF2 E Internal error(s) occurred.
CPF9541 E Not in debug mode.
CPF9542 E View not found.
CPF9549 E Error addressing API parameter.

Retrieve Debug Attribute (QteRetrieveDebugAttribute) API

Parameters

Required Parameter Group:

1	Debug attribute	Input	Char(10)
2	Attribute value	Output	Char(10)
3	Error code	I/O	Char(*)

The Retrieve Debug Attribute (QteRetrieveDebugAttribute) API is used to retrieve the attributes of the source debug session. These attributes may be any of the following:

- Default attributes established when the debug session was started
- Attributes changed with the Set Debug Attribute API
- Attributes changed by the Change Debug (CHGDBG) command

The attributes of the debug environment cannot be retrieved unless the job is currently in debug mode.

Required Parameter Group

Debug attribute
INPUT; CHAR(10)
The name of the debug environment attribute that is retrieved. The valid values for this parameter are:

***UPDPROD**
Retrieves the value of the update production files attribute.
***DEBUGJOB**
Retrieves an indicator of which job is being debugged.

Attribute value
OUTPUT; CHAR(10)
The current value of the attribute identified in the debug attribute parameter.

When the debug attribute contains *UPDPROD, the attribute value parameter can have one of the following values:

***YES** Allow the updating of production files while in debug mode.
***NO** Do not allow the updating of production files while in debug mode.

When the debug attribute parameter contains *DEBUGJOB, the attribute value parameter can have one of the following values:

***LOCAL**
The debug session is debugging programs that run in the job in which this API is running.
***REMOTE**
The debug session is debugging programs that run in the job specified in the Start Service Job (STRSRVJOB) 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.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Internal error(s) occurred.
CPF9541 E Not in debug mode.
CPF9559 E Debug attribute not defined.
CPF9549 E Error addressing API parameter.

Retrieve Module Views (QteRetrieveModuleViews) 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 name	Input	Char(20)
5	Program type	Input	Char(10)
6	Module name	Input	Char(10)
7	Returned library	Output	Char(10)
8	Error code	I/O	Char(*)

The Retrieve Module Views (QteRetrieveModuleViews) API is used to return a list of modules and views associated with a specified program to the caller of the API. The list includes both of the following:

- All modules bound to the program that can be debugged
- Every view (by number and type) that was created by the compiler when the module object was created

If you specify a module name, a list of views for that module is returned. If you specify *ALL for the module name, the list includes all modules for a given program.

Information returned by the Retrieve Module Views API is used by the calling program as input parameters to the Register Debug View API. Every module returned has at least one view associated with it. This is the statement view. It can be assumed that any additional views returned have text associated with them, and source debug can be done on these modules.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
A variable that is to receive the information requested. You can specify the size of this area to be smaller than that needed to hold the information. In this case, only part of the information is returned. However, the number of bytes that the API needs to return all of the information is still returned.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the receiver variable, the results are not predictable. The minimum length is 8 bytes.

It is suggested that a length of 8 be passed to the API, which fills in the first two fields of the receiver variable. One of the fields, bytes available, indicates how much space must be provided. This space can then be obtained, and a second call to the API can be made.

Format name
INPUT; CHAR(8)
The content and format of the module view information that is returned. The only valid value for this parameter is:

VEWL0100
Module view information. For more information, see "VEWL0100 Format" on page 12-7.

Program name
INPUT; CHAR(20)
The name of a program about which module and view information is listed. The first 10 characters contain the program name. The second 10 characters contain the name of the library where the program can be located. Both entries must be left-justified. The following special values may be used for the library name:

***CURLIB** The job's current library.
***LIBL** The library list.

Program type
INPUT; CHAR(10)
The type of program for which a view is to be registered. This is the object type of the program object. The allowable values are:

***PGM** Callable program
***SRVPGM** Service program

Module name
INPUT; CHAR(10)
A module name or *ALL (*ALL refers to all modules in the program).

Returned library
OUTPUT; CHAR(10)
The library where the program was found. This is useful when *LIBL or *CURLIB is specified for the program 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.

VEWL0100 Format

The following table shows the format of the receiver variable for the VEWL0100 format. For more information on the fields, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of elements
Note: The offsets to the following fields equal the offset to the last fixed-length field plus the length of the previous variable length fields.			
		CHAR(10)	Module name
		CHAR(10)	View type
		CHAR(20)	Compiler ID
		CHAR(10)	Main indicator
		CHAR(13)	View timestamp
		CHAR(50)	View description
		CHAR(3)	Reserved
		BINARY(4)	View number
		BINARY(4)	Number of views

All views for a module are listed together in the receiver variable. When the first view for a module is processed, it can be assumed that there is a total number of views for that module. The views are contiguous.

Field Descriptions

Bytes available. The number of bytes of data available to be returned to the user.

Bytes returned. The number of bytes of data returned to the user.

Compiler ID. The ID of the compiler that generated this view.

Main indicator. Whether the module is a main module (entry point) for the program. The main indicator field can have one of the following values:

**MAIN* Module is a main module
**NOMAIN* Module is not a main module

There is at most one main module per program. Service programs contain no main entry point.

Module name. The name of the module for this list entry.

Number of elements. The number of elements returned in the receiver variable. Each element has the same format, and it is repeated in the receiver variable. If the number of elements is zero and the receiver variable has room for at least one element, the program has no views in the module requested. If the module requested is *ALL, zero elements indicate the program cannot be debugged.

Number of views. The number of views in this module listed in the receiver variable.

Reserved. An ignored field.

View description. A character string that describes the view.

View number. A number that identifies a view within a module. Each view has a unique view number, which is used when you specify a specific view to register using the Register Debug View API.

View timestamp. The timestamp indicating when the view was created. It has the format of the American National Standard timestamp.

View type. The type of view. The view type can be one of the following values:

**TEXT*
: This is a view where text comes from files or text supplied by the processor.

**LISTING*
: This is a view where text comes entirely from text supplied by the processor.

statement
: This is a view consisting of statement identifiers. All modules have a statement view.

Error Messages

CPF3C21 E Format name not valid.
CPF3C24 E Receiver variable length error.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Internal error(s) occurred.
CPF9541 E Not in debug mode.
CPF954F E Module not found.
CPF955F E Program is not an ILE program.
CPF9801 E Object not found.
CPF9802 E Not authorized to object.
CPF9803 E Cannot allocate object.
CPF9809 E Library cannot be accessed.
CPF9810 E Library not found.
CPF9820 E Not authorized to library.
CPF9549 E Error addressing API parameter.

Retrieve Program Variable (QTERTVPV) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Program variable	Input	Char(132)
4	Basing pointer	Input	Array(5) of Char(132)
5	Start	Input	Binary(4)
6	Length	Input	Binary(4)
7	Output format	Input	Char(10)
8	Program	Input	Char(10)
9	Recursion level	Input	Binary(4)
10	Error code	I/O	Char(*)

The Retrieve Program Variable (QTERTVPV) API retrieves the current value of one program variable in a program that is being debugged. The information is returned to the calling program in a receiver variable. The amount of returned information is limited to the size of the receiver variable. This information is similar to the information returned using the Display Program Variable (DSPPGMVAR) command.

This API is valid only in debug mode and supports original program model (OPM) programs only. It cannot be used if the user is servicing another job and that job is on a job queue, or is held, suspended, or ended.

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The variable that is to receive the information requested. The minimum size for this area is 8 bytes. If the size of this area is smaller than the available information, the API returns only the data that the area can hold.

See "Format of Receiver Variable" on page 12-9 for details about the format.

Length of receiver variable

INPUT; BINARY(4)

The length of the receiver variable. If this value is larger than the actual size of storage allocated for the receiver variable, the results are not predictable. The minimum length is 8 bytes.

Program variable

INPUT; CHAR(132)

The name of the program variable whose value is to be retrieved. Possible values follow:

***CHAR** This special value is specified instead of a variable name if a basing pointer is also specified. This special value returns a character view of the area addressed by a pointer.

Program variable name

The name of the program variable. If the program variable is an array and it is specified with a subscript representing an element in the array, one array element is returned. If an array name is specified without any subscripts, all of the array elements are returned. The following can be specified for an array subscript:

- An integer
- Machine-interface object-definition-table-vector (MI ODV) number
- Asterisk (single-dimensional cross-section)
- A numeric variable name

Basing pointer

INPUT; ARRAY(5) of CHAR(132)

In languages where a program variable may be based on a pointer variable, you can specify the basing pointers for the variable to be retrieved. Up to five basing pointers may be specified. If the basing pointer is an element in an array, the subscript representing an element in the array must be specified. Up to 132 characters can be specified for one basing pointer name. If no basing pointer is specified, then the structure must be initialized to blanks. If one or more basing pointers are specified, then the subsequent array entries must be initialized to blanks. For more information on basing pointers, refer to the DSPPGMVAR command.

Start

INPUT; BINARY(4)

For string variables only, the starting position in the string from which its value is being retrieved. For a bit string, the value is the starting bit position. For a character string, the value is the starting character position.

Starting position

The retrieved value of the program variable from the starting position through the length specified in the length parameter.

This parameter is ignored on nonstring variables but must be initialized to any number greater than 0.

Length

INPUT; BINARY(4)

For string variables only, the length of the string retrieved, starting at the position specified by the start parameter. For a bit string, this value is the number of bits to retrieve. For a character string, this value is the number of characters to retrieve.

0 The value of the string variable is retrieved to the end of the string or retrieved for 200 bytes, whichever is less. If the string variable has a maximum length of zero, only 0 is allowed.

Retrieve length

The length of data to retrieve.

This parameter is ignored on nonstring variables but must be initialized to any number 0 or greater.

Output format
INPUT; CHAR(10)
The format in which the value is to be returned.

***CHAR** The value of the program variable is returned in character form.
***HEX** The value of the program variable is returned in hexadecimal form.

Program
INPUT; CHAR(10)
The name of the program that contains the program variable to be retrieved.

***DFTPGM** The program currently specified as the default program will be used.
Program name
The name of the program whose program variable is retrieved.

Recursion level
INPUT; BINARY(4)
The recursion level of the program that contains the program variable.

0 The last (most recent) call of the program is the one from which the automatic program variable is retrieved.
n The number of the recursion level of the program from which the automatic program variable is retrieved.

This parameter is ignored on static variables but must be initialized to any number 0 or greater.

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 Receiver Variable

The following table shows the information supplied in the receiver variable parameter. For more information on each field, see "Field Descriptions" on page 12-11.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Variable type
12	C	BINARY(4)	Data error
16	10	POINTER	Pointer to variable
32	20	BINARY(4)	Bit position
36	24	BINARY(4)	Variable length
40	28	BINARY(4)	Variable precision
44	2C	BINARY(4)	Number of array dimensions
48	30	BINARY(4)	Number of array elements returned
52	34	ARRAY(15) of BINARY(4)	Subscript bounds
172	AC	BINARY(4)	Element length
176	B0	BINARY(4)	Character string length
180	B4	CHAR(64)	Reserved
244	F4	CHAR(*)	Data retrieved

The following tables show the information supplied in the data retrieved field. The variable type field indicates which table is used.

Data for Binary Numeric (1)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(*)	Variable value

Data for Floating Point (2)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(*)	Variable value

Data for Zoned Decimal (3)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(*)	Variable value

Data for Packed Decimal (4)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(*)	Variable value

Data for Fixed Character (5)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(*)	Variable value

Data for Varying Character (6)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	BINARY(4)	Varying character length
256	100	CHAR(*)	Variable value

Data for Fixed Bit (7)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(*)	Variable value

Data for Unsigned Binary (8)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(*)	Variable value

Data for Space Pointer (9)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(8)	Hexadecimal offset
260	104	CHAR(8)	Reserved
268	10C	CHAR(30)	Object addressed by pointer
298	12A	CHAR(10)	Library name
308	134	CHAR(8)	Object type

Data for Data Pointer (10)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FA	CHAR(1)	Reserved
252	FB	CHAR(8)	Hexadecimal offset
260	104	CHAR(30)	Object addressed by pointer
290	122	CHAR(10)	Library name
300	12C	CHAR(8)	Object type
308	134	BINARY(4)	Data type
312	138	BINARY(4)	Data length
316	13C	BINARY(4)	Data precision
320	140	BINARY(4)	Data string length
324	144	BINARY(4)	Element length
328	148	CHAR(*)	Data

Data for Instruction Definition List (11)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(8)	Instruction number
260	104	CHAR(8)	Reserved
268	10C	CHAR(30)	Object addressed by pointer
298	12A	CHAR(10)	Library name
308	134	CHAR(8)	Object type

Data for System Pointer (12)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(16)	Authorization
268	10C	CHAR(30)	Object addressed by pointer
298	12A	CHAR(10)	Library name
308	134	CHAR(8)	Object type

Data for Machine Space Pointer (13)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(8)	Hexadecimal offset
260	104	CHAR(8)	Reserved
268	10C	CHAR(30)	Object addressed by pointer
298	12A	CHAR(10)	Library name

Offset			
Dec	Hex	Type	Field
308	134	CHAR(8)	Object type

Data for Exception Description (14)

Offset			
Dec	Hex	Type	Field
244	F4	CHAR(7)	Message ID
251	FB	CHAR(1)	Reserved
252	FC	CHAR(1)	Control
253	FD	CHAR(1)	Handler type
254	FE	CHAR(8)	Instruction number
262	106	CHAR(10)	Program name
272	110	CHAR(10)	Library name
282	11A	CHAR(2)	Reserved
284	11C	BINARY(4)	Compare string length
288	120	CHAR(28)	Compare string
316	13C	CHAR(1)	Job log
317	13D	CHAR(3)	Message type
320	140	BINARY(4)	Number of message IDs
324	144	ARRAY(*) of CHAR(7)	Array of messages

Field Descriptions

Array of messages. An array of the number of message IDs is returned.

Authorization. Pointer authorization.

Bit position. The starting bit position, 1–8, for bit strings returned in *HEX format. The least significant bit is 1 and 8 the most significant bit. This field will be initialized to 0 for any other variable type.

Bytes available. The number of bytes of data available to be returned to the user.

Bytes returned. The number of bytes of data returned to the user.

Character string length. For output format *CHAR, this value is the length of the returned character string. For output format *HEX, this value is initialized to 0. For fixed character, varying character, and fixed bit variables this field contains the actual length of the data returned for *CHAR and *HEX output formats. For pointers and exception monitors this field is 0.

Comparison string. The specified comparison string.

Comparison string length. The length of the comparison string. This value is 0 if a value is not specified.

Control. Exception monitor control action. The following values may be returned:

X'00' Default
X'01' Off
X'02' Resignal
X'04' Defer
X'05' Handle

Data. The data addressed by the pointer. This field is returned in the corresponding output format for the variable type (data type).

Data error. Whether an error was returned when returning a variable.

0 No errors were returned with the variable data.
1 One or more errors were returned with the variable data.

Data length. The length of the data addressed by the pointer. This is the same value as in the variable length field in the header.

Data precision. The precision of the data addressed by the pointer. This is the same value as in the variable precision field in the header.

Data retrieved. If an error is encountered while retrieving the data, CPD messages may be returned instead of the variable data. The structure of this parameter is dependent on the object type. The format of the data depends on the variable type field.

Data string length. The string length of the data addressed by the pointer. This is the same value as in the variable string length field in the header.

Data type. The type of data addressed by the pointer. This is the same value as in the variable type field in the header.

Element length. The length of the data element returned. If this field is 0, each element can be a different length and the user must go to the element to get the element length.

Handler type. Exception monitor handler type.

'00'X External handler
'01'X Call internal handler
'02'X Branch point handler

Hexadecimal offset. Hexadecimal offset of the space pointed to by the space or machine space pointer.

Instruction number. The exception handler instruction number for a monitor with an internal handler or X'0' for an external handler.

Job log. Put messages on job log.

0 No
1 Yes

Library name. The library containing the object addressed by the pointer, *LIBL, or X'0' for internal monitors.

Message ID. If an error was received with the variable data, this field contains the diagnostic message ID. If no error was received with the variable's data, this field contains blanks.

Message type. Message types being monitored.

100 Escape
010 Notify
001 Status

More than one message type can be monitored at a time. If the first and second characters are 1's, then escape and status messages are being monitored.

Number of array dimensions. If the variable is an array or an element of an array, this field is the number of array dimensions. Otherwise, this field is initialized to 0.

Number of array elements returned. If the variable is an array, this field is the number of array elements returned. Otherwise, this field is initialized to 0.

Number of message IDs. The number of message identifiers being monitored.

Object addressed by pointer. The fully qualified name of the object addressed by the pointer.

Object type. The Machine Interface (MI) type of the object addressed by the pointer.

Pointer to variable. Pointer to variable, if applicable. For example, a pointer is not returned to a variable of type machine space pointer or for an exception description. For system security reasons a pointer will not be returned if the security level is 50.

Program name. External handler program name or X'0' for an internal monitor.

Reserved. An ignored field.

Subscript bounds. The subscript lower bounds and subscript upper bounds for each array dimension. If the variable is not an element of an array, or the dimension is not used, the subscript lower and upper bounds are initialized to 0.

Varying character length. The actual length of the varying character string.

Variable length. The length of the variable data. For bit strings, this value is the number of bits. For packed and zoned variables, this value is the number of digits. For pointers and exception monitors this field is 0. For all other variable types, this value is the number of bytes.

Variable precision. The number of decimal digits or fractional digits for zoned and packed variables. For any other variable type, this field will be initialized to 0.

Variable type. The following are the possible variable types:

1 Binary numeric
2 Floating point
3 Zoned decimal
4 Packed decimal
5 Fixed character
6 Varying character
7 Fixed bit
8 Unsigned binary
9 Space pointer
10 Data pointer
11 Instruction definition list
12 System pointer
13 Machine space pointer
14 Exception description

Variable value. The value of the variable being retrieved. The following messages may be returned in this field:

CPD1901 Decimal data error
CPD1902 Pointer not set
CPD1903 Inexact result
CPD1904 Object not found
CPD1905 External data object not found
CPD1906 Pointer contained in destroyed object
CPD1907 Object suspended
CPD1908 Space addressing error
CPD1909 Pointer not 16-byte aligned
CPD1910 HLL pointer does not contain a valid pointer
CPD1911 Length plus start values exceeds variable length
CPD1913 Space addressing error (dumped in HEX)
CPD1914 Pointer references destroyed object

Error Messages

CPF1902 E No default program exists.
CPF1903 E Program not in debug mode.
CPF1905 E Value for starting position not valid.
CPF1906 E Command not valid.
CPF1915 E Value for length parameter not valid.
CPF1919 E Value for call level not valid.
CPF1927 E Special value for output format not valid.
CPF1938 E Serviced job not active.
CPF1939 E Serviced job timed out.
CPF1941 E Serviced job completed.
CPF24B4 E Severe error addressing parameter list.
CPF3C19 E Error with receiver variable.
CPF3C24 E Length of receiver variable not valid.
CPF7133 E Variable or basing pointer name missing.
CPF9872 E Parameter addressing error.
CPF9549 E Error addressing API parameter.

Retrieve Stopped Position (QteRetrieveStoppedPosition) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	View ID	Input	Binary(4)
4	Error code	I/O	Char(*)

The Retrieve Stopped Position (QteRetrieveStoppedPosition) API is used to determine if a module in a program is on the call stack. It indicates the position in the view at which the program stopped if the problem is on the stack. The caller must specify a registered view ID. If a procedure in the module indicated by the view ID is on the stack, a line number in the view is returned. This indicates that the view is stopped at that line.

If no procedure in the identified module is on the stack, a zero is returned.

The most recently called procedure in the specified module is the one whose line is returned. If a program is on the stack, the stack is searched from the most recent call backward until a procedure in the module is found. The location in that procedure is returned.

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. For more information, see "Format of Receiver Variable."

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.

It is suggested that a receiver variable length be given that is large enough to hold one position element. Because this normally is the number of elements that are returned, a single call to this API is usually sufficient. Also, this allows the number of elements field to be used to determine whether the program is stopped. If zero elements are returned, the program is not stopped in the specified view.

View ID
INPUT; BINARY(4)
The identifier of a previously registered view obtained using the Register Debug View 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.

Format of Receiver Variable

The following table shows the information supplied in the receiver variable parameter. For more information on the fields see, "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of stopped positions
Note: The following fields are repeated for each stopped position.			
*	*	BINARY(4)	Line number
*	*	BINARY(4)	Column number

Field Descriptions

Bytes available. The number of bytes of data available to be returned to the user.

Bytes returned. The number of bytes of data returned to the user.

Column number. The column number within the line number specified where the program is stopped in the view ID. Column numbers can be 1 through 255.

Line number. The line number within the view ID where the program is stopped. This number represents the line number within the view that corresponds to text retrieved using the Retrieve View Text API.

Number of stopped positions. A stopped position consists of the line number and column number fields and are repeated this number of times, once for each position available. If the view is not on the stack, this number is zero. If there is no room in the receiver variable to hold any stopped positions, this number is also zero. Therefore, there should be enough room in the receiver variable to hold at least one stopped position.

Because of program optimization, it is possible for the API not to know exactly where the view is stopped. For this reason, more than one position may be returned.

Error Messages

CPF3C24 E Receiver variable length not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Internal error(s) occurred.
CPF9541 E Not in debug mode.
CPF9542 E View not found.

| CPF9548 E Map not available.
| CPF9549 E Error addressing API parameter.

Retrieve View Text (QteRetrieveViewText) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	View ID	Input	Binary(4)
4	Start line number	Input	Binary(4)
5	Number of lines	Input	Binary(4)
6	Line length	Input	Binary(4)
7	Error code	I/O	Char(*)

The Retrieve View Text (QteRetrieveViewText) API is used to retrieve source text from a specified view. This text may be formatted and displayed by the user of this API. The caller must specify the following:

- A registered view ID
- The starting line number to be retrieved
- The number of lines of text to retrieve
- A buffer to contain the text retrieved

The source debugger support expands any include statements that are referred to when retrieving the source text associated with a view.

All text retrieved, whether it comes from files or as text supplied by a processor, is in the CCSID of the job.

If source files have changed since the view was created, diagnostic message CPF9561 is sent to the calling program's message queue for each file. Error CPF9566 also is issued, and all of the text available is retrieved. The calling program should warn the user that the view text may be incorrect.

If a source file cannot be accessed because it is deleted or the user is not authorized, error CPF9565 is issued. No more text is retrieved. Text up to that file is retrieved and this is indicated in the fields of the receiver variable. If the calling program attempts to read text in the view following the file, the starting line number can be set to a line after the file. The number of lines in the file that should have been read is returned in the exception data. This allows the calling program to skip over this file if desired.

It is suggested that the calling program buffer the retrieved text to minimize use of this API. Source files accessed by this API do not remain open across API calls. Performance degradation occurs for every use of the API that results in file access because of opening and closing files.

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. For more information, see "Format of Receiver Variable."

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable parameter. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

View ID
INPUT; BINARY(4)
The identifier of a previously registered view obtained by using the Register Debug View API.

Start line number
INPUT; BINARY(4)
The number of the first line that the API retrieved.

Number of lines
INPUT; BINARY(4)
The number of lines of source text to be retrieved. This number includes the line specified in the start line number parameter. Fewer than the number of lines may be placed in the receiver variable if fewer lines than requested are available. No more than the number of lines specified is placed in the receiver variable.

The following special value is supported for this parameter:

0 All of the text associated with this view should be retrieved.

Line length
INPUT; BINARY(4)
The length of each line of text to be retrieved. Each line takes exactly this many characters. If the actual line of text is shorter, it is padded to the right with blanks. If the line is longer than this length, it is truncated to fit. The line length must be a number from 1 through 255.

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 Receiver Variable

The following tables show the information supplied in the receiver variable parameter. The information returned depends on the type of view being used. For more information on each field, see "Field Descriptions" on page 12-15.

For the listing view:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of lines returned
12	C	BINARY(4)	Line length
Note: The following field is repeated for each line returned. The number of characters is equal to the line length.			
		CHAR(*)	Listing view source line

For the statement view:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of lines returned
12	C	BINARY(4)	Line length
Note: The following fields are repeated for each line returned. The total number of characters in each line is equal to the line length (10 + 10 + 10 + * = line length).			
		CHAR(10)	Procedure dictionary number
		CHAR(10)	Statement number
		CHAR(10)	Statement type number
		CHAR(*)	Procedure name

For the text view:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of lines returned
12	C	BINARY(4)	Line length
Note: The following fields are repeated for each line returned.			
		CHAR(12)	Sequence number
		CHAR(*)	Text view source line

Field Descriptions

Bytes available. The number of bytes of data available to be returned to the user.

Bytes returned. The number of bytes of data returned to the user.

Line length. The length of each line of text in the receiver variable parameter.

Listing view source line. The text associated with each line retrieved. The number of characters in each line is equal to the line length and there is no separation between lines.

Number of lines returned. The number of lines of source text retrieved by this API and available in the receiver variable. This may be less than the number of lines requested or available, if the receiver variable is not large enough to hold the text requested.

Procedure dictionary number. The number that uniquely identifies the procedure in this module. The number is left-justified and padded on the right with blanks.

Procedure name. The name of the procedure. The name is left-justified and padded on the right with blanks.

Sequence number. If the text is from a source physical file, these 12 bytes contain the sequence number for that line. If the text is supplied by the compiler, these 12 bytes are blank.

Statement number. The number that uniquely identifies the statement in the procedure. The number is left-justified and padded on the right with blanks. This number is shown on the compiler listing.

Statement type number. The type of statement produced by the compiler. The number is left-justified and padded on the right with blanks. Possible values are:

0 INIT CODE
1 PROC PROLOG
2 PROC EPILOG
3 BLKPENTRY
4 BLK EXIT
5 ALLOC
6 STMT
7 ENTRY
8 EXIT
9 MULTIEXIT
10 PATH LABEL
11 PATH CALL BGN
12 PATH CALL RET
13 PATH DO BGN
14 PATH TRUEIF
15 PATH FALSEIF
16 PATH WHEN BGN
17 PATH OTHERW
18 GOTO

Text view source line. The text associated with each line retrieved. The number of characters in each line equals the line length minus 12 bytes (the sequence number).

Error Messages

CPF3C24 E Receiver variable length not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Internal error(s) occurred.
CPF9541 E Not in debug mode.

CPF9542 E View not found.
CPF954A E No source text.
CPF9560 E Line length not valid.
CPF9561 E Source file has changed.
CPF9563 E Number of lines not valid.
CPF9564 E Starting line number not valid.
CPF9565 E Source cannot be accessed.
CPF9566 E One or more files have changed.
CPF9549 E Error addressing API parameter.

Start Source Debug (QteStartSourceDebug) API

Parameters

Required Parameter Group:

1	Program name	Input	Char(20)
2	Format name	Input	Char(8)
3	Error code	I/O	Char(*)

The Start Source Debug (QteStartSourceDebug) API lets you use the source debugging APIs in your session. This allows the debugging of any ILE programs or service programs that contain debug information.

Your job must be put in debug mode before this API is issued. Debug mode is a special environment in which the debug functions can be used in addition to routine system functions. Debug functions cannot be used outside debug mode. To start debug mode, you must issue the Start Debug (STRDBG) command.

The Start Source Debug API must be used before an ILE program can be debugged. This API requires that you specify a user exit program to be called by the source debugger support to handle breakpoints, steps, and unmonitored exceptions.

Your job remains in debug mode until an End Source Debug (EndSourceDebug) API is issued or until your current routing step ends.

If the job is servicing another job, the job will actually debug the job being serviced.

Required Parameter Group

Program name
INPUT; CHAR(20)
The name of the exit program that is called whenever a breakpoint, a program step, or an unmonitored exception occurs. See "Program-Stop Handler Exit Program" on page 12-24 for a discussion of the parameters passed to this program to assist in processing breakpoint, step, and exception information.

The first 10 characters contain the program name. The second 10 characters contain the name of the library where the program is located. Both entries must be left-justified.

Format name
INPUT; CHAR(8)
The content and format of the parameter list and data made available to the exit program specified in the program name parameter. The only valid value is:

STRD0100 The name of the parameter group belonging to the Program Stop Handler exit program. See, "Program-Stop Handler Exit Program" on page 12-24.

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.
CPF3CF2 E Internal error(s) occurred.
CPF9540 E Already in debug mode.
CPF9541 E Not in debug mode.
CPF9803 E Cannot allocate object.
CPF9809 E Library cannot be accessed.
CPF9810 E Library not found.
CPF9811 E Program not found.
CPF9820 E Not authorized to use library.
CPF9821 E Not authorized to program.
CPF9549 E Error addressing API parameter.

Submit Debug Command (QteSubmitDebugCommand) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	View ID	Input	Binary(4)
4	Input buffer	Input	Char(*)
5	Input buffer length	Input	Binary(4)
6	Compiler ID	Input	Char(20)
7	Error code	I/O	Char(*)

The Submit Debug Command (QteSubmitDebugCommand) API allows a client program to issue debug language statements. Debug language statements permit client programs to enter breakpoints, run one or more statements of the program under investigation (step), and evaluate expressions.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the results of the Submit Debug Command API. For more information on the structure of the receiver variable, see Figure 12-2 on page 12-18.

The Submit Debug Command API may have more data to return than can be stored in the receiver variable. The bytes available field, described in Figure 12-2 on page 12-18, specifies how large the receiver variable must be to contain the results for the Debug Language statements submitted. If more data is available than the receiver variable can contain, a larger buffer should be provided and the API should be reissued.

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.

View ID
INPUT; BINARY(4)
An identifier of a view of a module whose operation is managed by the source debugger. The view ID is returned as a result of issuing the Register Debug View API. The view ID is used to find debug data associated with the module.

Input buffer
INPUT; CHAR(*)
The input variable that is passed to the Submit Debug Command API. The information passed in the buffer is debug language statements.

Input buffer length
INPUT; BINARY(4)
The length of the data provided in the input buffer.

Compiler ID
INPUT; CHAR(20)
The language you should use in processing the debug command in the input 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.

Receiver Variable Format

The following table shows the structure of the receiver variable. For more information on the fields contained in the table, see "Field Descriptions."

Figure 12-1. Receiver Variable Structure

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
12	C	BINARY(4)	Entry count
16	10	CHAR(*)	Results array
		CHAR(*)	String space

Field Descriptions

Bytes available. The amount of space given to the API to store results. This field is assigned by the caller of the Submit Debug Command API.

Bytes returned. The amount of supplied space used if data was returned, or the amount of space needed if not enough space was made available. This field is assigned by the Submit Debug command API.

Entry count. The number of entries in the results array. This field is returned only if enough space was made available. The value of the field is the number of entries in the results array. Each entry occupies 12 bytes. Depending on the kind of information returned, values in entries vary.

Results array. The results of interpreting debug language statements. This is an array of records having similar structures. Each record in the array occupies 12 bytes. There can be up to three fields in each record. Each field occupies 4 bytes and can be interpreted as an unsigned (nonnegative) integer. The first field in a record is the result type field and is used to select the remaining fields. Entries in the result record array fall into several classes. Figure 12-2 on page 12-18 depicts several formats of result records.

Statements are interpreted sequentially and the results of each statement are placed in the order in which statements appear in the input buffer. The evaluate statement can return many values if an array or a structure is evaluated. The entry count field contains the number of entries in the results array, and the structure of each entry is summarized in Figure 12-2 on page 12-18.

String space. A sequence of strings. Each string is an array of characters whose last character is a null character.

Description of the Structure of the Receiver Variable: Figure 12-2 on page 12-18 illustrates three possible variations in the structure of the receiver variable. The receiver variable consists of the following structures:

- A header record

  This structure consists of three fields:

  - Bytes returned
  - Bytes available
  - Entry count

- A result array

- A string space

Figure 12-2. Variations in Receiver Variable Structure

Header	Bytes Returned	Bytes Available	Entry Count
Result array 1	Result type		
Result array 2	Result type	Count	
Result array 3	Result type	Offset	Length
String space			

Each row of Figure 12-2 occupies 12 bytes. The row containing the headings describes the remainder of the receiver variable. The number of bytes returned is assigned to that field. The value of the bytes returned field is always less than or equal to the size of the receiver variable. The number of bytes available may be greater than the number returned. In that case, the client program should reissue the Submit Debug Command API to obtain all data produced for the input debug language statements. The entry count field in the first row indicates the number of 12-byte records, each beginning with a result type field, that follow.

Records beginning with a result type field have the following basic formats.

- The first entry in the array shows a record containing only one field, result type. Records having this structure acknowledge that a kind of debug language statement was translated. An example of this kind of record is the result record for a step statement.
- The second entry in the array shows a record containing a count field as well as a result type field. The count field can serve two purposes:
  - It can acknowledge that a debug language statement was properly translated as in the case of the BreakR result record.
  - It can enumerate the number of related records to follow as in the case of the BreakpointR result record.
- The offset field contains the displacement from the start of the receiver variable to the first byte of the character string. All character strings are stored at the end of the receiver variable directly after the record entries. Displacements are measured in bytes.

  The length field contains the number of characters in the character string.

  The last character of each string in the string space has an ordinal value of zero. All characters in the string space occupy 8 bits. The length of a string in the string space does not include the last character.
- The last row of Figure 12-2 depicts an arbitrarily large string space containing character data. Names and other text fields that are referred to in the result type fields shown in the other rows of Figure 12-2 are stored in this area.

Results Array Entry Structure Summary

The following tables describe each result record in detail. Each result record contains up to three fields and always occupies 12 bytes. The first field, the result type field, is used as an enumerated type. The result type field determines the format of each result record.

Each of the following enumeration constants has both a symbolic name and an ordinal value. The terms ***symbolic*** and ***ordinal*** refer to enumerations found in programming languages. The symbolic value of an enumeration constant is the symbol, usually a descriptive word that serves as a keyword for the programmer (for example, StepR). The ordinal value of an enumeration constant is the integer constant assigned, usually by the compiler, to the symbolic value. For example, 1 is assigned for StepR.

StepR (1)

Record format StepR is returned as a result of evaluating a step statement.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Result type
4	4	BINARY(4)	Step count
8	8	CHAR(4)	Reserved

BreakR (2)

Record format BreakR contains the number of records returned for a break statement.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Result type
4	4	BINARY(4)	Break results count
8	8	CHAR(4)	Reserved

ClearBreakpointR (3)

Record format ClearBreakpointR contains the line number of the breakpoint removed as a result of interpreting the CLEARbreak-position statement.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Result type
4	4	BINARY(4)	Line number

Offset			
Dec	Hex	Type	Field
8	8	CHAR(4)	Reserved

ClearPgmR (4)

Record format ClearPgmR indicates that all breakpoints have been removed in the current program as result of interpreting the CLEAR PGM statement.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Result type
4	4	CHAR(4)	Reserved
8	8	CHAR(4)	Reserved

BreakPositionR (5)

Record format BreakPositionR identifies the line number on which a breakpoint was entered. This may not be the same line number as the one entered in the break statement.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Result type
4	4	BINARY(4)	Line number
8	8	CHAR(4)	Reserved

EvaluationR (6)

Record format EvaluationR contains the number of records returned for an evaluate statement that are referred to in the subsequent ExpressionValueR record.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Result type
4	4	BINARY(4)	Evaluation count
8	8	BINARY(4)	Reserved

ExpressionTextR (7)

Record format ExpressionTextR describes a character string that contains the expression that was evaluated by the evaluate statement.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Result type
4	4	BINARY(4)	Expression text offset
8	8	BINARY(4)	Expression text length

ExpressionValueR (8)

Record format ExpressionValueR refers to text that contains the formatted value of the expression that is described by the ExpressionTextR record.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Result type
4	4	BINARY(4)	Expression value offset
8	8	BINARY(4)	Expression value length

ExpressionTypeR (9)

Record format ExpressionTypeR contains the type of the expression whose value is referred to in the ExpressionValueR record.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Result type
4	4	BINARY(4)	Expression type
8	8	CHAR(4)	Reserved

QualifyR (10)

Record format QualifyR contains the line number entered as an operand in the qualify statement.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Result type
4	4	BINARY(4)	Line number
8	8	BINARY(4)	Reserved

Field Descriptions

Break results count. The number of entries returned for the break statement.

Expression text length. The number of characters in the expression text.

Expression text offset. The displacement from the start of the receiver variable to the first character of the expression text. Displacement is measured in bytes.

Expression value length. The number of characters in the expression value text.

Expression value offset. The displacement from the start of the receiver variable to the first byte of the expression value text. Displacement is measured in bytes.

Evaluation count. The number of records returned for an evaluate statement.

Expression type. The data type of the expression. The expression type may be one of the following values:

0 Error: An error occurred in evaluating the expression.
1 Char_8: An 8-bit (8 bits = 1 byte) character.
5 Card_32: A 32-bit unsigned integer.
7 Int_32: A 32-bit two's complement integer.
9 Real_64: A 64-bit IEEE 754 floating point value.
10 Pointer: 128-bit space pointer.
14 Enum_32: A 32-bit integer value.

Line number. The number of the line on which the action requested was performed.

Reserved. An ignored field.

Result type. The ordinal value of the result array.

Step count. The number of statements processed.

Statement Results

Break Statement Results. The Submit Debug Command API returns a detailed description of the break-position and conditional expression of a conditional breakpoint when a break statement is translated. The items returned follow:

- The number of records returned as a result of evaluating a break statement. Record type BreakR contains this information.
- The position on which the breakpoint was entered. Record type BreakPositionR contains the line number of the line on which the breakpoint was entered. Be aware that the input line number may be mapped to a different line. For example, a breakpoint entered on a line that contains a comment is mapped to the next line that contains an operational statement.
- The text of the expression that defines a conditional breakpoint. Record type ExpressionTextR refers to the text of the condition.

The break statement is interpreted. Program operation is managed by OS/400 according to the definition of the break statement.

Clear Statement Results. One record is returned. The record type depends on the operand following the keyword CLEAR. If the operand is a line number, the record type is ClearBreakpointR. If the operand is the keyword PGM, the record type is ClearPgmR.

The ClearBreakpointR record contains the line number input for the break-position.

The clear statement is interpreted. One or more breakpoints are removed from the program under investigation.

Evaluate Statement Results. An evaluate statement produces a variable number of Result Records. The first four result records follow:

- An EvaluationR record is returned, which enumerates itself and subsequent records. The EvaluationR result record always contains an evaluation count of four.
- An Expression text record is returned, which contains the offset and length of the string, which represents the expression text.
- An Expression value record is returned, which contains the offset and length of the string, which represents the value of the expression.
- An Expression type record is returned, which contains the type of the expression.

A single value is returned for an arithmetic expression or scalar variable. Multiple values are returned when a structure is evaluated. Refer to section "Submit Debug Command API Examples" on page A-53 for examples of the result records returned when a structure or an array is evaluated.

The evaluate statement is interpreted. Data is formatted according to the type of the input expression. Refer to Figure 12-3 on page 12-22 for a description of presentation formats.

Qualify Statement Results. One record is returned. The value of the result type field is QualifyR. The QualifyR record contains the input line number used to establish the current locality for subsequent evaluate statements.

A reference to the block that defines the current locality is assigned by the qualify statement.

Step Statement Results. One record is returned. The value of the result type field is StepR. The StepR record contains the number of statements to be run when control is given to the program under investigation.

The step statement is interpreted. Program processing is managed by OS/400 according to the definition of the step statement.

Error Messages

CPF1938 E Service job is held.
CPF1939 E Timeout waiting for serviced job.
CPF1941 E Service job has completed.
CPF3C19 E Error with receiver variable.
CPF3C24 E Receiver variable length not valid.
CPF3CF1 E Error code parameter not valid.
CPF7102 E Unable to add breakpoint or trace. Machine being serviced.
CPF7E01 E No receiver variable provided.
CPF7E02 E Receiver variable length not valid.
CPF7E03 E No input buffer provided.

| CPF7E04 E Input buffer length not valid.
| CPF7E05 E Truncated input buffer.
| CPF7E06 E No error code provided.
| CPF7E07 E Truncated error code.
| CPF7E08 E Error code not valid.
| CPF7E09 E Error code not valid.
| CPF7E10 E Internal error.
| CPF7E11 E Type error.
| CPF7E12 E Missing identifier.
| CPF7E14 E Missing field.
| CPF7E15 E Syntax error.
| CPF7E16 E Lexical error.
| CPF7E17 E Record type error.
| CPF7E18 E Pointer type error.
| CPF7E19 E Integral type error.
| CPF7E1A E Enumerated type error.
| CPF7E1B E Arithmetic type error.
| CPF7E1C E Scalar type error.
| CPF7E1D E Addressing error.
| CPF7E1E E Adding type error.
| CPF7E1F E Subtracting type error.
| CPF7E20 E Relational type error.
| CPF7E21 E Equality type error.
| CPF7E22 E Casting type error.
| CPF7E23 E Assignment type error.
| CPF7E24 E Line number not found.
| CPF7E25 E Array type error.
| CPF7E26 E Subscript type error.
| CPF7E27 E Display type error.
| CPF7E28 E Type error occurred.
| CPF7E29 E Bit type error.
| CPF7E2A E String constant error.
| CPF7E2B E Not compatible type.
| CPF8E03 E Internal error.
| CPF8E04 E Internal error.
| CPF8E05 E Error on equal operator.
| CPF8E06 E Error on not equal operator.
| CPF8E07 E Error on greater than operator.
| CPF8E08 E Error on greater than or equal operator.
| CPF8E09 E Error on less than operator.
| CPF8E0A E Error on less than or equal operator.
| CPF8E0B E Error on logical and operator.
| CPF8E0C E Error on logical or operator.
| CPF8E0D E Error on exclusive or operator.
| CPF8E0E E Error on logical not operator.
| CPF8E0F E Error on add operator.
| CPF8E10 E Error on subtract operator.
| CPF8E11 E Error on negate operator.
| CPF8E12 E Error on multiply operator.
| CPF8E13 E Error on divide operator.
| CPF8E14 E Error on increment operator.
| CPF8E15 E Error on decrement operator.
| CPF8E16 E Error on modulo operator.
| CPF8E17 E Pointer not set error.
| CPF8E18 E Conversion error not valid.
| CPF8E19 E Error on absolute value operator.
| CPF8E1A E Domain violation occurred.
| CPF8E1B E Domain violation occurred.
| CPF9541 E Not in debug mode.
| CPF9542 E View not found.
| CPF9549 E Error addressing API parameter.

| Debug Language Statements

| Debug language statements are the principal mechanism by
| which a programmer debugs a program. Programmers
| control the operation of a program by:

- | Entering break statements to select where the program will stop
- | Entering step statements to run one or more statements of the program under investigation

| The clear statement enables programmers to remove a par-
| ticular breakpoint or all breakpoints. Information about the
| state of the program being debugged can be extracted when
| program processing is suspended. The evaluate statement
| permits programmers to display the value of an expression,
| or to display an aggregate, and to alter the value of a vari-
| able.

| Debug language statements are constructed by the client
| program and placed in the input buffer. The Submit Debug
| Command API accepts the debug language statements of
| BREAK, CLEAR, EVAL, QUAL, and STEP.

| Break Statement

| The break statement permits a programmer to enter a break-
| point. Breakpoints are entered on the program under investi-
| gation. When the program under investigation encounters a
| breakpoint, operation is suspended.

| The following example shows a break statement:

```
| ─►BREAK──►position──►
| ─►BREAK──►position──┬──────────────────────┬►
|                     └─WHEN──►expression────┘
```

| The position marks where program operation is suspended
| when a breakpoint is encountered. Line numbers are used
| to identify the position when the break statement is entered.
| The line number entered is mapped to a statement by the
| Submit Debug Command API. A breakpoint causes the
| program to stop just before the break statement is run.

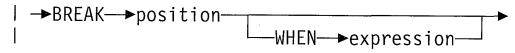
| Unconditional and conditional breakpoints can be entered.
| Unconditional breakpoints are discussed first, followed by a
| discussion of conditional breakpoints.

| An unconditional breakpoint is entered by issuing the first
| form of the break statement.

```
| ─►BREAK──►position──►
```

| A line number is entered for the position. Line numbers are
| associated with each view in that they identify the lines of
| source in a view. Line numbers are assigned sequentially
| beginning with line one.

| A conditional breakpoint is entered by issuing the second
| form of the break statement.



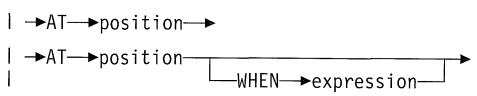
The position of a conditional breakpoint is assigned in the same way as the position in an unconditional breakpoint. A line number is entered for the position.

The condition of a conditional breakpoint is the expression following the reserved word WHEN. The result of the expression must have a Boolean or a logical value when evaluated. The expression is interpreted before the statement on which the breakpoint was entered is run. If the value of the expression is TRUE, operation of the program investigation is suspended. If the value of the expression is FALSE, operation continues without interruption.

The locality of variables used in the conditional expression is defined by the line number that defines the position.

A breakpoint can be replaced by entering another breakpoint using the same position. The most recent breakpoint entered on a position is the active breakpoint.

BREAK may be replaced by the reserved word AT in the statement that defines the break statement.

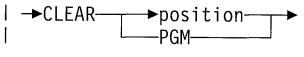


Clear Statement

The clear statement enables a programmer to remove a particular breakpoint or all breakpoints in a program. The first example of the clear statement allows programmers to delete a particular breakpoint. Breakpoints are identified by the number of the line on which they were entered. A breakpoint is removed by issuing the clear statement with the line number that identifies the breakpoint.

All breakpoints in a program are removed when the second example of the clear statement is issued.

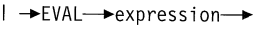
The following example shows a clear statement:



Evaluate Statement

The expression appearing in an evaluate statement is evaluated, and the value of the expression is returned. The value of an expression is formatted according to the expression type.

The following example shows an evaluate statement:



An evaluate statement allows a programmer to display the value of an expression or an aggregate, or to alter the value of a variable. The precise definition of what can be displayed or altered is dependent on the language of the module being debugged.

Variables can be displayed or altered when program processing is suspended. Program operation is temporarily suspended as a result of encountering a breakpoint or completing a step statement.

Variables are formatted according to their type recorded in the HLL symbol table, and according to the language of the module being debugged. Formats available include integer, hexadecimal, exponential, and address, among others. The format of the variable indicates the type of the variable in C-language modules.

The locality of variables that appear in an evaluate statement is defined by the most recently run qualify statement. The program calling this API is advised to issue a qualify statement that defines the stop position when program operation is suspended.

EVAL may be replaced by the reserved word LIST in the statement that defines the evaluate statement.

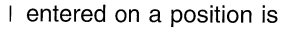


Figure 12-3 describes the formatting of data by type.

Figure 12-3. Presentation Formats

Type	Format	Example
Char_8	'c'	'A'
Int_32	-dd...d +dd...d	+546
Real_64	+d.d...dE+dd -d.d...dE-dd	-1.2345678901234E-95
Pointer	SPP:xx xx xx xx xx xx	SPP:COFEA100 1200
Enum_32	cc...c	Yellow (2)

Locality: Locality is the term used to describe the range over which an entity may be referred to in a module. The terms locality and scope are synonymous. By this definition, the locality of an entity is always confined to the compilation unit in which it was declared.

Entity is a formal way of describing all things that can be declared in a module. Variables, procedures, labels, types, and constants are entities.

The locality of an entity is defined by the block in which it is declared. An entity is visible in the block in which it is declared and all subordinate blocks. A variable can be referred to in the block in which it is declared.

An entity may be declared in a block that encloses other blocks. The entity declared in the outer, enclosing block is visible in inner blocks if the name does not collide with other entities in inner blocks. A variable declared in an outer block can be referred to in an inner block if no variable of the same name is declared in the inner block.

To fully qualify a particular locality in a program, both program and module must be identified.

Qualify Statement

The qualify statement permits a programmer to define the locality of variables that appear in succeeding evaluate statements. Locality is defined by the line number operand on the qualify statement. The locality assigned is that block in which the line number appears.

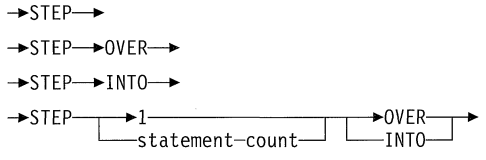
The following example shows a qualify statement:

The locality assigned when a qualify statement is issued remains in effect until the next qualify statement is issued. The Submit Debug Command API keeps the locality assigned for the purpose of evaluating expressions. Users of the Submit Debug Command API are advised to issue the qualify statement whenever program operation is suspended. Use the line number of the stopped position to identify the current locality. In this way, programmers may issue several evaluate statements that refer to variables that are defined in the locality of the stopped position.

Step Statement

The step statement permits a programmer to run one or more statements of the program under investigation for testing purposes. The program being tested runs the number of statements specified in the statement-count operand. Operation of the program under test is suspended at completion of the step statement.

The following example shows a step statement:



If no value is entered for the statement-count, one statement is run.

The reserved words OVER and INTO direct the source debugger to step over or into procedures, respectively. If OVER appears in a step statement, the source debugger does not suspend operation in any procedures that are called. Procedures and functions are run without interruption.

The INTO reserved word directs the source debugger to stop in procedures that are called.

If neither INTO or OVER is entered on the step statement, OVER is assumed.

Step statement limitations follow:

- Procedures in modules having no debug data cannot be entered using the step statement.
- Procedures in other programs cannot be entered using the step statement.

Debug Session Handler Exit Program

Parameters

Required Parameter Group:

1	Reason	Input	Char(10)
2	Program list	Input	Char(*)
3	Number of programs	Input	Binary(4)

The Debug Session Handler exit program is a user-written program that manages the Integrated Language Environment (ILE) debugger. It determines when the debugger starts, stops, and shows its displays.

The name of the program is specified in the SRCDBGPGM parameter of the Start Debug (STRDBG) command. This program is called by the STRDBG command to initialize the user-written debugger, and is called by the End Debug (ENDDBG) command to end it. It is also called by the STRDBG and the Display Module Source (DSPMODSRC) commands to show the Display Module Source display.

Required Parameter Group

Reason
INPUT; CHAR(10)
The reason the program was called. Valid reasons include:

***START**
The program-stop handler should be initialized by the Start Source Debug API.

***STOP**
The program-stop handler should be removed by the End Source Debug API.

***DISPLAY**
The debugger should display itself.

Program list
INPUT; CHAR(*)
The list that is to receive a list of ILE programs to add to the debugger. This list contains the number of program entries, each entry being 30 characters in length. The first 10 characters contain the name of the program. The second 10 characters contain the name of the library where the program is located. The third 10 characters contain the type of object being named and can be *PGM (a callable program) or *SRVPGM (a service program). Each name is left-justified within the field. This parameter is only valid if the Reason parameter is *START.

Number of programs
INPUT; BINARY(4)
The number of programs stored in the program list parameter. This parameter is only valid if the reason parameter is *START.

Program-Stop Handler Exit Program

Parameters

Required Parameter Group:

1	Qualified program name	Input	Char(20)
2	Program type	Input	Char(10)
3	Module name	Input	Char(10)
4	Stop reason	Input	Char(10)
5	Receiver variable	Input	Char(*)
6	Number of entries	Input	Binary(4)
7	Message data	Input	Char(*)

The Program-Stop Handler exit program is a user-written program that handles program-stop conditions.

This program must be identified to the Source Debugger support with the Start Source Debug (QteStartSourceDebug) API.

The location at which the program-stop condition occurred is specified in the receiver variable parameter and is in terms of the statement view. The user-supplied program may use the Map View Position (MapViewPosition) API to determine the location to which this program maps any other registered view.

This user exit program is called by the debugger support when the format name registered on the Start Source Debug (StartSourceDebug) API is STRD0100.

Unmonitored exceptions are reported through this exit program only when the program and module identified have been created. Without debugging data, the message that is the cause of the unmonitored exception is issued, and the Program-Stop Handler exit program is not called.

Required Parameter Group

Qualified program name
INPUT; CHAR(20)
The name of the program that is stopped as a result of a breakpoint, program step, or unmonitored exception.

The first 10 characters contain the name of the program. The second 10 characters contain the name of the library where the program is located. Each name is left-justified.

Program type
INPUT; CHAR(10)
The object type of the program that is stopped. The possible values are:

***PGM** Bound program
***SRVPGM** Service program

Module name
INPUT; CHAR(10)
The name of the module (left-justified) that is stopped.

Stop reason
INPUT; CHAR(10)
The reason the program was called. Each character of this parameter has a specific meaning. The characters and their meanings are:

1 This reason is set when an unmonitored exception is received by the program being serviced by the source debugger support.
0 No unmonitored exception
1 Unmonitored exception

2 The program stopped because an unconditional or conditional breakpoint was satisfied.
0 No break condition
1 Break condition

3 The program stopped because a step condition was reached.
0 No step condition
1 Step condition

4 The program stopped because a conditional breakpoint was set and there was a failure in running the condition. The program is stopped at the break position specified.
0 No break condition failure
1 Break condition failure

5-10 Reserved. These characters are set to 0.

Receiver variable
INPUT; CHAR(*)
A list of locations within the statement view where the program stop condition occurred. This list contains the number of entries where each number is defined as follows:

Line number
BINARY(4)
The line number in the statement view defined by the dictionary number of the procedure and the HLL statement number within that procedure where the program is stopped.

Number of entries
INPUT; BINARY(4)
The number of positions stored in the receiver variable parameter. In some cases, it is not known exactly where a program is stopped; therefore, multiple positions are given. Each entry specifies one position in the statement view. This number is not less than one nor greater than three. At least one stopped position will be identified; if stopped at more than one position, no more than the first three positions are made available.

Message data
INPUT; CHAR(*)
Information about the message. The information in this parameter is valid only when the stop reason specified is an unmonitored exception. For a detailed description of the format, see "Format of Message Data" on page 12-25.

Format of Message Data: The following table shows the information supplied in the message data parameter. For more information on the fields, see "Field Descriptions" on page 12-25.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Length of message data
4	4	CHAR(7)	Message ID
11	B	CHAR(20)	Message file
31	1F	CHAR(1)	Reserved
32	20	CHAR(512)	Message data

Field Descriptions

Length of message data. The length of the data available in the message data parameter, in bytes. This field contains the length of the available message data for the predefined message indicated in the message ID field.

Message data. The values for substitution variables in the predefined message specified in the message ID field and located in the message file field.

Message file. The name of the message file that contains the message identified in the message ID field.

The first 10 characters contain the message file name. The second 10 characters contain the name of the library where the file can be located. Both entries are left-justified.

Message ID. The identifying code of the message.

Reserved. An ignored field.

Part 6. Dynamic Screen Manager APIs

Chapter 13. Dynamic Screen Manager APIs Introduction

The Dynamic Screen Manager (DSM) APIs are a set of screen I/O interfaces that provide a dynamic way to create and manage screens for the Integrated Language Environment (ILE) high-level languages. Because the DSM interfaces are bindable, they are accessible to ILE programs only.

The DSM APIs provide an alternative to the existing way of defining screen appearance outside a program by coding in DDS or UIM, for example. Instead, programmers can use a series of calls to DSM within their programs to dynamically specify and control screen appearance for their applications. Unlike static definition methods, the DSM interfaces provide the flexibility needed for those applications requiring more dynamic screen control. The DSM support provided varies from low-level interfaces for direct screen manipulation to windowing support.

The DSM APIs fall into the following functional groups:

- **Low-level services**

  The low-level services APIs provide a direct interface to the 5250 data stream commands. These APIs are used to query and manipulate the state of the screen; to create, query, and manipulate input and command buffers used to interact with the screen; and to define fields and write data to the screen.

- **Window services**

  The window services APIs are used to create, delete, move, and resize windows, and to manage multiple windows during a session.

- **Session services**

  The session services APIs provide a general scrolling interface that can be used to create, query, and manipulate sessions, and to perform input and output operations to sessions.

Each group of DSM services APIs is discussed in detail in the chapters that follow.

Data Structures for DSM APIs

Data structures for use with the DSM APIs are available in the QUSRTOOL library for the ILE high-level languages. For example, to use the ILE C/400 data structures in your programs, copy the members found in source physical file QATTSYSC in library QUSRTOOL into your own file and library and rename the members as follows:

QATTSYSC Member Name	Rename Member to:
OPSN1API	QSNLL.H
OPSN2API	QSNWIN.H
OPSN3API	QSNSESS.H

The mnemonics referred to throughout the DSM chapters are contained in these files.

Omitting Parameters with Associated Lengths

To omit a parameter with an associated length parameter, that length parameter should be omitted or specified as 0. If the length parameter is specified with a value greater than 0, the parameter with which it is associated is required.

For example, to omit the user extension information on the low-level environment, specify either a NULL pointer by value, or 0 by reference for the length. The extension information structure is ignored. If the length is greater than 0, the extension information structure cannot be NULL. If it is, then a `Required parameter omitted` error is generated.

Chapter 14. Low-Level Screen I/O Services APIs

The low-level screen I/O services APIs provide a direct interface to the 5250 data stream commands and user-defined data streams for performing basic screen I/O operations. For detailed information about the results and interactions of these operations, refer to the following 5250 data stream documentation:

- *5250 Functions Reference*, SA21-9247
- *5394 Remote Control Unit Functions Reference*, SC30-3488
- *5494 Remote Control Unit Functions Reference*, SC30-3533

The low-level services are divided into the following functional groups:

- **Screen Manipulation and Query APIs** are used to query and manipulate the state of the screen.
- **Buffer Manipulation and Query APIs** are used to create, query, and manipulate input and command buffers used to interact with the screen.
- **Screen Input APIs** are used to read field data and other information from the screen.
- **Screen Output APIs** are used to define fields and write data and other information to the screen.

Error Handling

Calls to most of the interfaces can result in a direct I/O operation, or in the storing of commands in a command buffer. The command buffer provides a way of saving the commands so that multiple operations can be specified and performed in a single I/O operation. DSM performs error handling as much as possible prior to issuing an I/O operation. For example, if a request is made to place the screen in wide mode, and the display does not support this mode, DSM detects and reports the error condition before performing an I/O operation. This way of handling errors is particularly useful in the case where multiple commands have been saved in a buffer. Otherwise, there is no obvious way to determine which command was in error when the I/O operation fails.

The errors that can occur for each operation are listed with the operation. If an error message indicates that the error is issued for a negative response code, this means that the error was not detected by DSM and occurred on the I/O operation.

Device Support

The 5250 Query command is used to determine the valid commands for a particular device. This command is issued for the current device at the start of each DSM session and the information is saved for subsequent queries. If the 5250 Query command is not supported, the base data stream support as documented in the *5250 Functions Reference* is assumed, with color and wide support being determined by the device type.

Operating Environments

The low-level interfaces operate within an environment that can be defined using the Create Low-Level Environment (QsnCrtEnv) API. The low-level environment defines the operating modes, such as DBCS support and the window mode. The environment is passed as a parameter to most of the low-level services APIs. There is no need to define a low-level environment unless you need a specific operating environment that is different from the default. The default low-level environment is indicated on the low-level service APIs by specifying the environment handle as zero.

Direct and Indirect Operations

Many of the low-level APIs accept an optional command buffer as a parameter. For such APIs, the command buffer can be used to store and accumulate a group of requested operations. The accumulated operations can then be written to the screen in a single I/O operation. Better performance can be achieved because a group of repetitive operations can be issued to the screen without having to re-invoke the sequence of individual APIs for each repeated operation.

A **direct operation** is one that omits the command buffer. The requested operation takes place immediately. For most APIs, specifying the command buffer results in an **indirect operation**. No I/O operation takes place and the operation is simply stored in the command buffer. Several of the screen input APIs, however, do perform a direct operation when the command buffer is specified. This semantic is discussed in "Read Input Fields (QsnReadInp) API" on page 17-5.

DBCS Considerations

You can write DBCS data enclosed with SO/SI to the screen, and when the underlying display supports it, graphic DBCS using the Write Data (QsnWrtDta) API to specify the data stream Write Extended Attribute order. You can define fields as being DBCS through the Set Field (QsnSetFld) API using the appropriate field control word. DBCS data can be written

to such fields as described above. If you specify DBCS support on the low-level environment description, (see "Format of the Low-Level Environment Description" on page 15-4), the APIs will handle the parsing of DBCS data and fields appropriately.

The APIs do not provide any special processing of DBCS data, such as adding SO/SI to DBCS graphic data when graphic data is not supported by the display. For example, if you want to define a field as graphic DBCS and write graphic DBCS data to it, code the QsnSetFld API specifying a control word of DSM_FCW_DBCS_PURE (x'8220') and then write the graphic data to a command buffer using the QsnWrtDta API. Precede and follow this data with Write Extended Attribute orders to add the extended NLS SO/SI attributes. If you want to write a graphic data value to a non-graphic DBCS field, you must enclose the graphic DBCS data with SO/SI prior to calling the QsnWrtDta API.

5250 Data Stream Details

AID-Generating Keys

The AID (attention indicator) code identifies to the host system the function being requested from the keyboard. The AID code is returned via certain input operations when the operator presses an AID-generating key. Figure 14-1 lists the AID-generating keys and the AID codes associated with each key. See "Format of the Low-Level Environment Description" on page 15-4 for instructions on how to specify an alternative help key.

Figure 14-1. AID Codes

AID key	Mnemonic	AID Code
Cmd 1 - 12 (cmd 1=x'31', cmd12=x'3C')	QSN_F1 - QSN_F12	x'31' - x'3C'
Selector Light Pen Auto Enter	QSN_SLP	x'3F'
Forward Edge Trigger Auto Enter	QSN_FET	x'50'
PA1	QSN_PA1	x'6C'
PA2	QSN_PA2	x'6E'
PA3	QSN_PA3	x'6B'
Cmd 13 - 24 (cmd 13=x'B1', cmd24=x'BC')	QSN_F13 - QSN_F24	x'B1' - x'BC'
Clear	QSN_CLEAR	x'BD'
Enter or Record Advance	QSN_ENTER	x'F1'
Help (not in error state)	QSN_HELP	x'F3'
Roll Down or Page Up	QSN_ROLLDOWN or QSN_PAGEUP	x'F4'
Roll Up or Page Down	QSN_ROLLUP or QSN_PAGEDOWN	x'F5'
Print	QSN_PRINT	x'F6'
Record Backspace	QSN_RECBS	x'F8'

Control Characters

The display control characters (CCs) are always specified as a pair of 1-byte fields. They are used on the QsnWTD, QsnReadInp, QsnReadMDT, and QsnReadMDTAlt APIs. These characters select specific operations for the display station to perform. Byte 1 is always processed first. When the CCs are used with the QsnWTD API, the first CC is processed immediately while the second CC is not processed until all the other information associated with the API has been processed. When used with an input operation, both CCs are processed after the operation has completed. Figure 14-2 and Figure 14-3 on page 14-3 list the valid control character values and their associated mnemonics.

Figure 14-2 (Page 1 of 2). Control Character Byte 1

Mnemonic	Bits 0-2	Reset Pending Aid; Lock Keyboard	Clear Master MDT;Reset MDT Flags in Nonbypass Fields	Clear Master MDT;Reset MDT Flags in All Fields	Null Nonbypass Fields with MDT On	Null All Nonbypass Fields
QSN_CC1_NULL	000					
QSN_CC1_LOCKBD	001	x				
QSN_CC1_MDTNBY	010	x	x			

Figure 14-2 (Page 2 of 2). Control Character Byte 1

Mnemonic	Bits 0-2	Reset Pending Aid; Lock Keyboard	Clear Master MDT;Reset MDT Flags in Nonbypass Fields	Clear Master MDT;Reset MDT Flags in All Fields	Null Nonbypass Fields with MDT On	Null All Nonbypass Fields
QSN_CC1_MDTALL	011	x		x		
QSN_CC1_CLRMOD	100	x			x	
QSN_CC1_MDTNBY_CLRALL	101	x	x			x
QSN_CC1_MDTNBY_CLRMOD	110	x	x		x	
QSN_CC1_MDTALL_CLRALL	111	x		x		x

Note:

1. Bits 3 through 7 are reserved and must be 0. A CPFA31C error will be issued if this is not the case.
2. If there are no bypass fields with MDT flags on, then the master MDT will be cleared.

Figure 14-3. Control Character Byte 2

Mnemonic	Bit	Meaning
	0	reserved
QSN_CC2_NO_IC	1	0: Cursor moves to default or IC order position when keyboard unlocks 1: Cursor does not move when keyboard unlocks
QSN_CC2_RST_CSR_BL	2	0: no action 1: Reset blinking cursor
QSN_CC2_SET_CSR_BL	3	0: no action 1: Set blinking cursor
QSN_CC2_UNLOCKBD	4	0: no action 1: Unlock the keyboard and reset any pending AID bytes
QSN_CC2_ALARM	5	0: no action 1: Sound alarm
QSN_CC2_MSG_OFF	6	0: no action 1: Set Message Waiting indicator off
QSN_CC2_MSG_ON	7	0: no action 1: Set Message Waiting indicator on

Notes:

- The mnemonics for control character byte 2 can be combined with a bitwise OR operation.
- See notes in the 5250 data stream documentation for further details regarding these functions.

Screen Attribute Characters

The screen or field attributes control the image produced on the display station screen. Each attribute occupies one character position in the display station regeneration buffer and is displayed as a blank. The effect produced by an attribute begins at its location in the regeneration buffer and continues until the next attribute appears. The attributes for non-color displays are shown in Figure 14-4 and for color displays in Figure 14-5 on page 14-4. There are certain operations that allow a value to be specified for a screen attribute that indicates no screen attribute should be used. Where supported, the value is X'00' and the mnemonic is QSN_NO_SA.

Figure 14-4 (Page 1 of 2). Screen Attributes for Non-Color Displays

Mnemonic	Bits	Value
QSN_SA_NORM	0-2	001: Attribute identification flag

Figure 14-4 (Page 2 of 2). Screen Attributes for Non-Color Displays

Mnemonic	Bits	Value
QSN_SA_CS	3	0: Column separator off 1: Column separator on
QSN_SA_BL	4	0: Do not blink field 1: Blink field
QSN_SA_UL	5	0: Do not underscore field 1: Underscore field
QSN_SA_HI	6	0: Low intensity 1: High intensity
QSN_SA_RI	7	0: Normal image 1: Reverse image
QSN_SA_ND		Non-display: equivalent to specifying QSN_SA_UL, QSN_SA_HI, and QSN_SA_RI.
Note: Multiple functions can be selected by combining the mnemonics with a logical OR operation.		

Figure 14-5 (Page 1 of 2). Screen Attributes for Color Displays

Mnemonic	Value	Meaning
QSN_SA_GRN	x'20'	Green
QSN_SA_GRN_RI	x'21'	Green/Reverse Image
QSN_SA_WHT	x'22'	White
QSN_SA_WHT_RI	x'23'	White/Reverse Image
QSN_SA_GRN_UL	x'24'	Green/Underscore
QSN_SA_GRN_UL_RI	x'25'	Green/Underscore/Reverse Image
QSN_SA_WHT_UL	x'26'	White/Underscore
QSN_SA_ND	x'27'	Nondisplay
QSN_SA_RED	x'28'	Red
QSN_SA_RED_RI	x'29'	Red/Reverse Image
QSN_SA_RED_BL	x'2A'	Red/Blink
QSN_SA_RED_RI_BL	x'2B'	Red/Reverse Image/Blink
QSN_SA_RED_UL	x'2C'	Red/Underscore
QSN_SA_RED_UL_RI	x'2D'	Red/Underscore/Reverse Image
QSN_SA_RED_UL_BL	x'2E'	Red/Underscore/Blink
QSN_SA_ND_2F	x'2F'	Nondisplay
QSN_SA_TRQ_CS	x'30'	Turquoise/Column Separators
QSN_SA_TRQ_CS_RI	x'31'	Turquoise/Column Separators/Reverse Image
QSN_SA_YLW_CS	x'32'	Yellow/Column Separators
QSN_SA_YLW_CS_RI	x'33'	Yellow/Column Separators/Reverse Image
QSN_SA_TRQ_UL	x'34'	Turquoise/Underscore
QSN_SA_TRQ_UL_RI	x'35'	Turquoise/Underscore/Reverse Image
QSN_SA_YLW_UL	x'36'	Yellow/Underscore
QSN_SA_ND_37	x'37'	Nondisplay
QSN_SA_PNK	x'38'	Pink
QSN_SA_PNK_RI	x'39'	Pink/Reverse Image
QSN_SA_BLU	x'3A'	Blue
QSN_SA_BLU_RI	x'3B'	Blue/Reverse Image
QSN_SA_PNK_UL	x'3C'	Pink/Underscore

Figure 14-5 (Page 2 of 2). Screen Attributes for Color Displays

Mnemonic	Value	Meaning
QSN_SA_PNK_UL_RI	x'3D'	Pink/Underscore/Reverse Image
QSN_SA_BLU_UL	x'3E'	Blue/Underscore
QSN_SA_ND_3F	x'3F'	Nondisplay

Display Address

The display address is the address at which data is displayed or a field definition begins. This can be modified explicitly with a Set Output Address (QsnSetOutAdr) call, or implicitly with output operations, such as those associated with the Write Data (QsnWrtDta) API, that accept a cursor position. The 5250 Write to Display (WTD) command initializes the display address to row 1, column 1. Since each output operation contains a WTD command, this means that the display address is reset on each direct screen output operation.

Insert Cursor Address

The insert cursor (IC) order specifies the position of the cursor when the host system unlocks the keyboard and when the display station operator presses the Home key. The display address is not affected by this address. This can be set with the Insert Cursor (QsnInsCsr) API, and in some cases with the Set Cursor Address (QsnSetCsrAdr) API (only when the Move Cursor (MC) order is not supported).

Modified Data Tag (MDT) Bit

There is a modified data tag (MDT) bit for each input field and a master MDT bit. These bits are used to determine which fields should be returned in response to the Read Modified Fields (QsnReadMDT), Read Modified Alternate (QsnReadMDTAlt), and Read Modified Immediate Alternate (QsnReadMDTImmAlt) APIs. The MDT bit for a field and the master MDT bit can be set using bit 4 of the field format word (see "Format of the Field Format Word" on page 18-12) on a Set Field (QsnSetFld) API. The master MDT bit and the MDT bit for a field are set on anytime the operator types into or alters a field on the display. Once the bits are set, only a control character for resetting them (see Figure 14-2 on page 14-2), or a clear screen operation using the Clear Screen (QsnClrScr) API or a Start of Header order, can reset them.

Resequencing

Resequencing allows the control unit to return up to 128 input fields in any specified order. Resequencing is accomplished by chaining input fields together with Field Control Words specifying resequencing. (See "Format of the Field Control Word" on page 18-15 and the 5250 data stream documentation for details.)

States and Modes

The display station can be in one of several states (conditions), each with its accompanying modes (methods of operation). The following is a list of these states and their associated modes:

- Hardware error state
- Normal locked state
- Normal unlocked state
  - Command mode
  - Insert mode
  - Data mode
- Power-on state
- Prehelp error state
- Post-help error state
- System services (SS) message state
- System request state

See the 5250 data stream documentation for a detailed explanation of each state and mode.

Chapter 15. Screen Manipulation and Query APIs

The screen manipulation and query APIs are used to query and manipulate the state of the screen and comprise the following:

- **Change Low-Level Environment** (QsnChgEnv) changes the low-level environment description.
- **Clear Field Table** (QsnClrFldTbl) clears the format table but does not alter the display.
- **Clear Screen** (QsnClrScr) clears the screen and sets the screen size.
- **Create Low-Level Environment** (QsnCrtEnv) creates the environment for the low-level services APIs.
- **Delete Low-Level Environment** (QsnDltEnv) deletes the environment for the low-level services APIs.
- **Initialize Low-Level Environment Description** (QsnInzEnvD) initializes the low-level environment description.
- **Query Color Support** (QsnQryColorSup) determines if the display device supports color.
- **Query Display Mode Support** (QsnQryModSup) queries if the display device allows a given mode.
- **Query 5250** (QsnQry5250) returns the results of the 5250 Query command.
- **Restore Screen** (QsnRstScr) restores the contents of a screen saved with Save Screen (QsnSavScr) API.
- **Retrieve Display Mode** (QsnRtvMod) queries the current device mode.
- **Retrieve Low-Level Environment Description** (QsnRtvEnvD) retrieves the low-level environment description.
- **Retrieve Low-Level Environment User Data** (QsnRtvEnvDta) returns a pointer to the user data for the low-level environment.
- **Retrieve Low-Level Environment Window Mode** (QsnRtvEnvWinMod) retrieves the low-level interface environment window mode description.
- **Retrieve Screen Dimensions** (QsnRtvScrDim) retrieves the screen dimensions.
- **Roll Down** (QsnRollDown) rolls the screen down by the given number of rows.
- **Roll Up** (QsnRollUp) rolls the screen up by the given number of rows.
- **Save Screen** (QsnSavScr) saves the present display so it can be restored later.
- **Set Low-Level Environment Window Mode** (QsnSetEnvWinMod) sets the window mode for the low-level interface environment.

The detailed API descriptions are presented in alphabetical order.

Change Low-Level Environment (QsnChgEnv) API

Parameters

Required Parameter Group:

1	Low-level environment description	Input	Char(*)
2	Length of low-level environment description	Input	Binary(4)

Optional Parameter Group:

3	Low-level environment handle	Input	Binary(4)
4	Error Code	I/O	Char(*)

Returned Value:

Return code	Output	Binary(4)

The Change Low-Level Environment (QsnChgEnv) API changes the description for the given low-level environment. The Change Low-Level Environment exit routine will be called if specified on the user extension information of the Create Low-Level Environment (QsnCrtEnv) API.

Required Parameter Group

Low-level environment description
INPUT; CHAR(*)
The new environment description for the given environment. The format of this parameter is shown in "Format of the Low-Level Environment Description" on page 15-4.

Length of low-level environment description
INPUT; Binary(4)
The length of the low-level environment description parameter. The value specified must be 16.

Optional Parameter Group

Low-level environment handle
INPUT; BINARY(4)
The low-level environment to be modified. If this parameter is omitted or specified as 0, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA327 E Low-level environment description value incorrect.
CPFA334 E Low-level environment handle incorrect.

Clear Field Table (QsnClrFldTbl) API

Parameters

Optional Parameter Group:

1	Command buffer handle	Input	Binary(4)
2	Low-level environment handle	Input	Binary(4)
3	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Clear Field Table (QsnClrFldTbl) API erases the contents of the format table without affecting the display station screen. This allows for example, all input field definitions to be erased from the screen without altering the physical appearance of the screen.

This command corresponds directly to the 5250 Clear Format Table command. See the 5250 data stream documentation for details.

Optional Parameter Group

Command buffer handle
INPUT; BINARY(4)
If this parameter is omitted or specified as 0, this is a direct operation and the format table is cleared immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place. If this parameter is omitted or specified as 0, this is a direct operation and the screen is cleared immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place. If this parameter is omitted or specified as 0, this is a direct operation and the screen is cleared immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Clear Screen (QsnClrScr) API

Parameters

Optional Parameter Group:

1	Mode	Input	Char(1)
2	Command buffer handle	Input	Binary(4)
3	Low-level environment handle	Input	Binary(4)
4	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Clear Screen (QsnClrScr) API clears the screen and sets the screen size to the specified mode. This command corresponds directly to the 5250 Clear Unit or Clear Unit Alternate command, depending upon the current screen

presentation size. See the 5250 data stream documentation for details.

Restrictions

If this is an indirect operation, it must be the first command in the command buffer.

Optional Parameter Group

Mode
INPUT; CHAR(1)
The mode to place the screen in after the screen is cleared. If this parameter is omitted, a value of 0 is assumed. The possible values are:

- **0** Indicates that the current screen size should be kept. For indirect operations where this value is specified, the subsequent clear operation will be based on the current screen size, not on whatever size the screen is when the command buffer is ultimately written out. The current display size will be determined using the QsnRtvMod interface.
- **3** Set screen to 24x80 mode.
- **4** Set screen to 27x132 mode. This value is not supported by all devices. A CPFA306 error will occur if an attempt is made to specify this value with a device that does not support it.

Command buffer handle
INPUT; BINARY(4)
If this parameter is omitted or specified as 0, this is a direct operation and the screen is cleared immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA306 E Command not supported by current device.
CPFA321 E Operation not first command in command buffer.
CPFA322 E Incorrect display mode &1 specified.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Create Low-Level Environment (QsnCrtEnv) API

Parameters

Required Parameter Group:

1	Low-level environment description	Input	Char(*)
2	Length of low-level environment description	Input	Binary(4)

Optional Parameter Group:

3	User Extension Information	Input	Char(*)
4	Length of user extension information	Input	Binary(4)
5	Low-level environment handle	Output	Binary(4)
6	Error Code	I/O	Char(*)

Returned Value:

	Low-level environment handle	Output	Binary(4)

The Create Low-Level Environment (QsnCrtEnv) API creates an operating environment for low-level interface routines.

Required Parameter Group

Low-level environment description
INPUT; CHAR(*)
The environment description for the low-level interfaces. The format of this parameter is shown in "Format of the Low-Level Environment Description" on page 15-4.

Length of low-level environment description
INPUT; Binary(4)
The length of the low-level environment description parameter. The value specified must be 16.

Optional Parameter Group

User extension information
INPUT; CHAR(*)
The user extension information is used to associate data and exit routines with the low-level environment. This essentially enables the object-oriented programming concept of inheritance, allowing the low-level environment to be extended in a natural way. The user extension information cannot be changed once the environment has been created. The format of this parameter is shown in "Format of the Low-Level User Environment Extension Information."

User extension information length
INPUT; BINARY(4)
The length of the user extension information parameter. If this parameter is specified with a zero value, the user extension information parameter is ignored. If a non-zero value is specified, it must be 48 and the user extension parameter is required.

Low-level environment handle
OUTPUT; BINARY(4)
The low-level environment that is created as a result of this operation.

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.

Returned Value

Low-level environment handle
OUTPUT; BINARY(4)
The value for parameter 5 is returned. If an error occurs during processing, returns -1.

Format of the Low-Level Environment Description

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Color support
1	1	CHAR(1)	Character conversion
2	2	CHAR(1)	CDRA conversions to 0x3F
3	3	CHAR(1)	DBCS support
4	4	CHAR(1)	Coexistence
5	5	CHAR(1)	Alternative help key support
6	6	CHAR(10)	Target device

Format of the Low-Level User Environment Extension Information

Offset		Type	Field
Dec	Hex		
0	0	PTR(SPP)	User data associated with the environment
16	16	PTR(PP)	Exit routine to call when the low-level environment is changed
32	32	PTR(PP)	Exit routine to call when the low-level environment is deleted

Field Descriptions

In the following descriptions, specifying the value *Same* indicates the current value when using the Change Low-Level Environment (QsnChgEnv) API. The default value refers to the value set by the Initialize Low-Level Environment Description (QsnInzEnvD) API.

Alternative help key support. Specifies if the alternative help key is used. The default is no alternative help key support.

The possible values for this field are:

0 Same: Keep the current setting.
1 None: No alternative help key support is used.
QSN_F1 through QSN_F24
The specified key is the alternative help key. See Figure 14-1 on page 14-2 for the values that correspond to these mnemonics. When this key is pressed, the AID code for the Help key will be returned.

CDRA conversions to X'3F'. When CDRA conversion takes place, all characters not supported in the target CCSID are converted to X'3F'. Sending data containing X'3F' to the display causes adverse effects.

This field specifies whether the DSM low-level routines are to check for X'3F' in the data to be displayed and perform any conversions if necessary. Conversion will be performed for both direct and indirect operations on data output through the QsnWrtDta and QsnWrtTDta APIs, and input through the QsnReadInp, QsnReadMDT, QsnReadMDTAlt, QsnReadMDTAlt, QsnReadImm, and QsnReadMDTImmAlt APIs.

The default is convert if character conversion (see below) is specified as convert. Otherwise, the default is standard. (See "Limitations and Restrictions" on page 18-29 for further details.) The possible values for this field are:

0 Same: Keep the current setting.
1 Standard: Always display data as standard data with no replacement.
2 Convert: Always check for X'3F' in data and convert to X'1F' before displaying the data and convert X'1F' in incoming data to X'3F'.

Character conversion. This field specifies whether CDRA conversion takes place on the data when the job CCSID

does not match that of the display device. Conversion will be performed for both direct and indirect operations on data output through the QsnWrtDta and QsnWrtTDta APIs, and input through the QsnReadInp, QsnReadMDT, QsnReadMDTAlt, QsnReadMDTAlt, QsnReadImm, and QsnReadMDTImmAlt APIs.

The CCSID for the display device is determined from the CHRID of the device. The possible values for this field are:

0 Same: Keep the current setting.
1 Standard: Do not perform conversion.
2 Convert: If the job CCSID does not match that of the display device and neither has the value 65535, perform the appropriate conversion on outgoing and incoming data. This is the default. (See "Limitations and Restrictions" on page 18-29 for further details.)

Coexistence. Whether DSM coexists with other screen I/O methods, such as DDS- or UIM-coded interfaces, during the course of this application. Better performance can be achieved if coexistence is not required; the DSM APIs can assume the state of the device, for example, wide or normal mode. The default is coexist.

The possible values for this field are:

0 Same: Keep the current setting.
1 Coexist: Other screen I/O methods are used in conjunction with DSM.
2 Only: DSM is the only screen I/O method being used.

Color support. This field determines the color selection used when both a monochrome and a color display attribute are supplied. The default is select.

The possible values for this field are:

0 Same: Keep the current setting.
1 Mono: Always use the monochrome attributes specified regardless of the underlying device type.
2 Color: Always use the color attributes specified regardless of the underlying device type.
3 Select: Select the appropriate attribute based on the underlying display type.

DBCS support. This field specifies whether the data being sent to the display contains DBCS data. This field affects, for example, how data is handled within sessions. For devices that do not support DBCS data, the default is standard; for DBCS-capable devices, it is mixed. If a value other than standard or mixed is specified for a device that does not support DBCS data, a CPFA306 error will occur.

The possible values for this field are:

0 Same: Keep the current setting.
1 Standard: Always handle data as single-byte characters.
2 Only: Data contains double-byte characters only. SO/SI control characters must enclose the data; these are not implicitly added.
3 Either: Data contains either double-byte or single-byte characters, but not both. SO/SI control characters must enclose the DBCS part of the data, unless a graphic DBCS value is passed. In this case, extended ideographic attributes must enclose the data, which can be written using the QsnWrtDta API (see "Write Data (QsnWrtDta) API" on page 18-17).
4 Mixed: Data may contain both single- and double-byte characters. All double-byte character strings within the data must be enclosed by SO/SI control characters. Graphic DBCS data must be enclosed by extended ideographic attributes as described for the preceding value.

Exit routine to call when low-level environment changed. Exit routine to call when the low-level environment is changed through the QsnChgEnv or QsnSetEnvWinMod API. For a description of the parameters passed to this routine, see "Change Low-Level Environment Exit Routine." Specify NULL for this field if no exit routine is required.

Exit routine to call when low-level environment deleted. Exit routine to call when the low-level environment is deleted through the QsnDltEnv API. The exit routine will be called before the environment itself is deleted. For a description of the parameters passed to this routine, see "Delete Low-Level Environment Exit Routine" on page 15-6. Specify NULL for this field if no exit routine is required.

User data associated with the environment. A pointer to any data the user wants to associate with this environment. This field can be used by the programmer to attach information to the low-level environment that can be of any format. For example, if multiple environments are being used, a list of the fields currently defined in an environment could be associated with the environment through this pointer. Specify NULL for this field if you do no have any user data.

Target device. The program device name of the target display device for the environment. This parameter must be specified with a value of *REQUESTER, which is the default.

Change Low-Level Environment Exit Routine

This exit routine, if specified on the user extension information, is called after the environment is changed through the QsnChgEnv or QsnSetEnvWinMod API. The following parameter is passed to the exit routine:

Parameter Passed to Exit Routine

1	Environment handle	Input	Binary(4)

Change Low-Level Environment Exit Routine Parameter

Low-level environment handle
INPUT; BINARY(4)
A handle for the environment that was changed.

Delete Low-Level Environment Exit Routine

This exit routine, if specified on the user extension information, is called before the environment is deleted. The following parameter is passed to the exit routine:

Parameter Passed to Exit Routine

1	Environment handle	Input	Binary(4)

Delete Low-Level Environment Exit Routine Parameter

Low-level environment handle
INPUT; BINARY(4)
A handle for the environment that was deleted.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA314 E Memory allocation error.
CPFA31E E Required parameter &1 omitted.
CPFA327 E Low-level environment description value incorrect.

Delete Low-Level Environment (QsnDltEnv) API

Parameters

Required Parameter:			
1	Low-level environment handle	Input	Binary(4)
Optional Parameter:			
2	Error Code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Delete Low-Level Environment (QsnDltEnv) API deallocates an operating environment for low-level interface routines.

Required Parameter

Low-level environment handle
INPUT; BINARY(4)
The low-level environment to be deallocated.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA318 E Error calling exit routine.
CPFA317 E Cannot deallocate memory dynamically.
CPFA31E E Required parameter &1 omitted.
CPFA334 E Low-level environment handle incorrect.

Initialize Low-Level Environment Description (QsnInzEnvD) API

Parameters

Required Parameter Group:			
1	Low-level environment description	Output	Char(*)
2	Length of low-level environment description	Input	Binary(4)
Optional Parameter:			
3	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Initialize Low-Level Environment Description (QsnInzEnvD) API initializes a low-level environment description with default values. Unless otherwise specified in the low-level environment description in "Format of the Low-Level Environment Description" on page 15-4, pointer fields are set to the null pointer, numeric fields to 0, character flag fields to 0, and other character fields to blanks. For example, the default value for the color support field is 3, so this field will be set to 3.

Required Parameter Group

Low-level environment description
OUTPUT; CHAR(*)
The low-level environment description to be initialized.

Length of low-level environment description
INPUT; Binary(4)
The length of the low-level environment description parameter. This parameter must be specified as 16.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.

Query Color Support (QsnQryColorSup) API

Parameters

Optional Parameter Group:

1	Color indication	Output	Char(1)
2	Low-level environment handle	Input	Binary(4)
3	Error code	I/O	Char(*)

Returned Value:

	Color indication	Output	Binary(4)

The Query Color Support (QsnQryColorSup) API determines whether the current display device supports color or not.

Optional Parameter Group

Color indication
OUTPUT; CHAR(1)
Whether the device supports color or not. This information will be set based on the results of the 5250 Query command, if the display supports it; otherwise, certain defaults are assumed. See "Device Support" on page 14-1 for details. The possible values are:

0 Device does not support color
1 Device supports color

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Color indication
OUTPUT; BINARY(4)
This API returns the value for the color indication parameter if successful, or -1 otherwise.

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.
CPFA334 E Low-level environment handle incorrect.

Query Display Mode Support (QsnQryModSup) API

Parameters

Required Parameter:

1	Display mode	Input	Char(1)

Optional Parameter Group:

2	Mode indication	Output	Char(1)
3	Low-level environment handle	Input	Binary(4)
4	Error code	I/O	Char(*)

Returned Value:

	Mode indication	Output	Binary(4)

The Query Display Mode Support (QsnQryModSup) API determines if the current display device supports the given mode. Certain devices, like the 3486 and 3487, support 27x132 mode but can be switched by keystroke to turn off the wide capability. This will be reflected in the result

returned by the QsnQryModSup API. Use this API to determine if a subsequent mode change request through the Clear Screen (QsnClrScr) API is valid. You can use the result of the Query 5250 (QsnQry5250) API to determine if the display is capable of supporting wide mode or not.

Required Parameter

Display mode
INPUT; CHAR(1)
The display mode for which to query support. The possible values are:

3 24x80 mode
4 27x132 mode

Optional Parameter Group

Mode indication
OUTPUT; CHAR(1)
Whether the device allows the specified mode or not. The possible values are:

0 Device does not support the mode
1 Device supports the mode

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Mode indication
OUTPUT; BINARY(4)
This API returns the value for the mode indication parameter if successful, or -1 otherwise.

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.
CPFA322 E Incorrect display mode &1 specified.
CPFA334 E Low-level environment handle incorrect.

Query 5250 (QsnQry5250) API

Parameters

Required Parameter Group:			
1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
Optional Parameter:			
3	Error Code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Query 5250 (QsnQry5250) API is used to retrieve the results of the Query 5250 command for the current device. The Query 5250 command returns device and controller attributes for the current device, such as whether wide mode and graphical user interface (GUI) are supported.

Restrictions

This command is not supported by all control units. A query status of 3 indicates if the query failed.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The receiver variable that is to receive the result of the query. 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. The format of the data returned is shown in "Format of the Query Data" on page 15-9.

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 unpredictable. The minimum length is 8 bytes.

The API returns as much information as it can fit in this length. If the available information is longer, it is truncated. If the available information is shorter, the unused output 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. To determine how much information the API could return if space were available, see the bytes available field.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Query Data

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(1)	Query status
9	9	BINARY(2)	Work station control unit
11	B	CHAR(3)	Code Level
14	E	CHAR(16)	Reserved field.
30	1E	CHAR(1)	Work station type code
31	1F	CHAR(4)	Machine type code
35	23	CHAR(3)	Model number
38	26	CHAR(1)	Keyboard ID
39	27	CHAR(1)	Extended keyboard ID
40	28	CHAR(1)	PC keyboard ID
41	29	CHAR(4)	Serial number
45	2D	BINARY(2)	Maximum input fields
47	2F	CHAR(2)	Control unit customization
49	31	CHAR(1)	Reserved field.
50	32	CHAR(12)	Device capabilities

Field Descriptions

Further details on the fields listed can be found in the *5494 Remote Control Unit Functions Reference* manual.

Bytes available. The number of bytes of data available to be returned.

Bytes returned. The number of bytes of data returned.

Code Level. Identifies the code release level.

Control unit customization. Indicates customization parameters for the control unit as:

Byte 0

- Bit 0: Indicates that the AS/400 system can send a 5250 WSC Customization command when set on
- Bit 1: Indicates that the AS/400 system can send a 5250 Query Station State command when set on
- Bits 2–7: Reserved

Byte 1: Reserved

Device capabilities. Defines the operating capabilities of the designated LU as:

Byte 0

- Bits 0–1: Indicate Row 1/Column 1 support as:
  B'00' no support
  B'01' limited support
- Bit 2: Indicates the Read MDT Alternate command is supported when set on
- Bit 3: Indicates the work station and control unit have PA1 and PA2 support when set on
- Bit 4: Indicates the work station and control unit have PA3 support when set on
- Bit 5: Indicates the work station and control unit have cursor select support when set on
- Bit 6: Indicates the work station and control unit have move cursor order support when set on
- Bit 7: Indicates the Read Modified Immediate Alternate command is supported when set on

Byte 1—display screen capabilities

- Bits 0–3: Define screen size as:
  B'0001' 24 x 80
  B'0011' 24 x 80 or 27 x 132
- Bit 4: Indicates selector light pen (SLP) is supported when set on
- Bit 5: Indicates magnetic stripe reader (MSR) is supported when set on
- Bits 6–7: Define color support as:
  B'00' Monochrome display
  B'01' Color support

Byte 2

- Bit 0: Indicates Text Symbols support when set on
- Bit 1: Indicates work station and control unit have extended primary attribute
- Bits 2–4: Indicate Office Editor/Text support as:
  B'000' No Office Editor/Text support
  B'001' single language Office Editor/Text support
  B'010' dual language Office Editor/Text support
- Bit 5: Indicates work station and control unit have extended primary attribute support in data processing (DP) mode (WEA order) when set on
- Bits 6–7: Indicates extended foreground color attribute support
  B'01' Available in DP mode. Fourteen colors are defined, but only seven are available. The other seven colors are mapped into the available colors.
  B'10' Available in DP mode. Fourteen colors are supported.

Byte 3

- Bits 0–2: Indicate ideographic capability as:

B'000' No ideographic capability
B'001' Ideographic capability for presentation screen only
B'010' Ideographic data type and presentation screen ideographic capability

- Bits 3–5: Indicate bidirectional support as:
  B'000' No bidirectional capability
  B'001' Bidirectional capability
- Bits 6: Ideographic
- Bits 7: Reserved

Byte 4

- Bits 0–2: Indicate graphics capability as:
  B'000' No graphics capability
  B'001' 5292-style graphics
  B'010' GDDM-OS/2 Link Graphics
- Bit 3: Indicates extended 3270 data stream capability when set on
- Bit 4: Indicates a pointer device is available when set on
- Bit 5: Indicates that GUI-like characters are available when set on
- Bit 6: Indicates the control unit supports enhanced user interface commands and field control words (FCWs) when set on.

  The commands include:
  - Create Window
  - Unrestricted Cursor Movement
  - Remove GUI Window
  - Remove All GUI Constructs
  - Read Screen To Print
  - Read Screen To Print With Extended Attributes
  - Write Error Code To Window
  - Save Partial Screen
  - Restore Partial Screen
  - Define Selection Field
  - Remove GUI Selection Field
  - Define Scroll Bar
  - Remove GUI Scroll Bar

  The FCWs include:
  - Continued
  - Cursor Progression
  - Highlighted
  - Pointer Device Selection.
- Bit 7: Indicates Write Error Code To Window command is supported when set on

Byte 5

- Bit 0: Indicates the Write Data and Programmable Mouse Buttons structured field commands, the Word Wrap FCW, and Ideographic Continued entry fields are supported when set on
- Bit 1: Indicates this is a GUI device which will use all-points-addressable constructs for windows, selection fields, and scroll bars, when set on
- Bits 2–7: Reserved

Bytes 6–11: Reserved

Extended keyboard ID. The device code for extended 5250 keyboards.

Keyboard ID. Reserved. This field is set to X'00'.

Machine type code. An EBCDIC code for the machine type.

Maximum input fields (2 bytes). The maximum number of input fields available (256).

Model number (3 bytes). An EBCDIC code for the machine model number.

PC keyboard ID. Device code for PC keyboards attached to a 5250 work station (X'00' for nonprogrammable work stations).

Query status. The status of the 5250 query data. The possible values are:

DSM_5250Q_YES (1)	Query information successfully retrieved.
DSM_5250Q_NO (2)	Query cannot be issued for the device. This occurs when the device configuration specifies that the query command should not be issued against the device.
DSM_5250Q_FAIL (3)	Query command failed. Default values are supplied based on the device type. This occurs, for example, when the controller does not support the query command.

Serial number. Field for device serial number. This field is set to zero for a work station with no serial number.

Work station control unit. The type of control unit.

Work station type code. The workstation type. The value is X'01' for display station.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C24 E Length of the receiver variable is not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.

Restore Screen (QsnRstScr) API

Parameters

Required Parameter:

1	Input buffer containing saved data	Input	Binary(4)

Optional Parameter Group:

2	Command buffer handle	Input	Binary(4)
3	Low-level environment handle	Input	Binary(4)
4	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Restore Screen (QsnRstScr) API restores the state of the display as saved with an indirect command. The display will be restored using the data contained in the input buffer given by parameter 1. If an indirect operation is specified, the resulting command buffer will contain the Restore Screen command and the data to restore the screen. Additional commands can be added to the command buffer subject to the conditions described in "Restrictions."

This command corresponds directly to the 5250 Restore Screen or Restore Partial Screen command. See the 5250 data stream documentation for details.

Restrictions

This command must be the last command in the command buffer except when GUI support is used. In this case, other input commands may follow.

Required Parameter

Input buffer containing saved data
INPUT; BINARY(4)
An input buffer that contains the result of an indirect QsnSavScr operation. The data will be copied from this input buffer and used for the restore screen operation.

Optional Parameter Group

Command buffer handle
INPUT; BINARY(4)
If this parameter is omitted or specified as 0, this is a direct operation and the screen is restored immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA316 E Saved data not valid.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Retrieve Display Mode (QsnRtvMod) API

Parameters

Optional Parameter Group:

1	Display mode	Output	Char(1)
2	Low-level environment handle	Input	Binary(4)
3	Error code	I/O	Char(*)

Returned Value:

	Display mode	Output	Char(1)

The Retrieve Display Mode (QsnRtvMod) API returns the current display mode.

Optional Parameter Group

Display mode
OUTPUT; CHAR(1)
The current display mode. The possible values are:

3 Device is in 24x80 mode

4 Device is in 27x132 mode

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Display mode
OUTPUT; CHAR(1)
This API returns the value for the display mode parameter, or 0 if an error occurs during processing.

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.
CPFA322 E Incorrect display mode &1 specified.
CPFA334 E Low-level environment handle incorrect.

Retrieve Low-Level Environment Description (QsnRtvEnvD) API

Parameters

Required Parameter Group:			
1	Low-level environment description	Output	Char(*)
2	Length of low-level environment description	Input	Binary(4)
Optional Parameter Group:			
3	Low-level environment handle	Input	Binary(4)
4	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Retrieve Low-Level Environment Description (QsnRtvEnvD) API returns the description corresponding to the specified low-level environment.

Required Parameter Group

Low-level environment description
Output; CHAR(*)
The variable that contains the low-level environment description when the QsnRtvEnvD API has completed. The format of the data returned is shown in "Format of the Data Returned."

Length of low-level environment description
INPUT; Binary(4)
The length of the low-level environment description parameter. The minimum length is 8. If the length is larger than the size of the receiver variable, the results are not predictable. The API returns as much information as it can fit in this length. It the available information is longer, it is truncated. If the available information is shorter, the unused output 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. To determine how much information the API could return if space were available, see the bytes available field.

Optional Parameter Group

Low-level environment handle
INPUT; BINARY(4)
The low-level environment for which the description should be retrieved. If this parameter is omitted or specified as 0, the description for the default low-level environment is retrieved.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Data Returned

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(*)	Environment description The format of the remaining data returned is shown in "Format of the Low-Level Environment Description" on page 15-4.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C24 E Length of the receiver variable is not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.
CPFA334 E Low-level environment handle incorrect.

Retrieve Low-Level Environment User Data (QsnRtvEnvDta) API

Parameters

Required Parameter:			
1	Low-level environment handle	Input	Binary(4)
Optional Parameter Group:			
2	User data pointer	Output	PTR(SPP)
3	Error code	I/O	Char(*)
Returned Value:			
	User data pointer	Output	PTR(SPP)

The Retrieve Low-Level Environment User Data (QsnRtvEnvDta) API returns a pointer to the user data for the given low-level environment.

Required Parameter

Low-level environment handle
INPUT; BINARY(4)
A handle for the low-level environment for which the user data should be returned. If this parameter is omitted or specified as 0, the default low-level environment is used.

Optional Parameter Group

User data pointer
OUTPUT; PTR(SPP)
A pointer to the user data, as specified on the low-level environment description, for the given low-level environment.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

User data pointer
OUTPUT; PTR(SPP)
This API returns the value for the user data pointer parameter, or the null pointer if an error occurs.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1F E Pointer parameter is not on a 16-byte boundary.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.
CPFA334 E Low-level environment handle incorrect.

Retrieve Low-Level Environment Window Mode (QsnRtvEnvWinMod) API

Parameters

Required Parameter Group:			
1	Window mode description	Output	Char(*)
2	Length of window mode description	Input	Binary(4)
Optional Parameter Group:			
3	Low-level environment handle	Input	Binary(4)
4	Error Code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Retrieve Low-Level Environment Window Mode (QsnRtvEnvWinMod) API queries the state of the window mode for the low-level interfaces.

Required Parameter Group

Window mode description
OUTPUT; CHAR(*)
The field in which the window mode description should be stored. The format of the data returned in this field is described in "Format of the Data Returned" on page 15-14.

Length of window mode description
INPUT; BINARY(4)
The length of the window mode description parameter. If the length is larger than the size of the receiver variable, the results are not predictable. The minimum length is 8. The API returns as much information as it can fit in this length. If the available information is longer, it is truncated. If the available information is shorter, the unused output 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. To determine how much information the API could return if space were available, see the bytes available field.

Optional Parameter Group

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Data Returned

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(1)	Window mode
9	9	CHAR(*)	Window mode description

Field Descriptions

Bytes available. The number of bytes of data available to be returned.

Bytes returned. The number of bytes of data returned.

Window mode. Whether window mode is enabled or disabled. The possible values are:

0 Window mode is disabled.
1 Window mode is enabled.

Window mode description. The format of the remaining data returned is shown in "Format of the Window Mode Description" on page 15-19.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.
CPFA334 E Low-level environment handle incorrect.

Retrieve Screen Dimensions (QsnRtvScrDim) API

Parameters

Optional Parameter Group:

1	Number of rows	Output	Binary(4)
2	Number of columns	Output	Binary(4)
3	Low-level environment handle	Input	Binary(4)
4	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Retrieve Screen Dimensions (QsnRtvScrDim) API retrieves the current dimensions of the screen. You must specify either the number-of-rows or the number-of-columns parameter, or a CPFA31E message will be issued.

Optional Parameter Group

Number of rows
OUTPUT; BINARY(4)
The current height of the screen. This information will be set based on the current display size.

Number of columns
OUTPUT; BINARY(4)
The current width of the screen. This information will be set based on the current display size.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA334 E Low-level environment handle incorrect.

Roll Down (QsnRollDown) API

Parameters

Required Parameter Group:

1	Number of lines to roll	Input	Binary(4)
2	Top row of roll area	Input	Binary(4)
3	Bottom row of roll area	Input	Binary(4)

Optional Parameter Group:

4	Command buffer handle	Input	Binary(4)
5	Low-level environment handle	Input	Binary(4)
6	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Roll Down (QsnRollDown) API rolls the screen down a given number of lines within the roll area specified. The following conditions cause a CPFA315 error to occur:

- A top row of zero
- A bottom row greater than the number of display lines
- A top row greater than or equal to the bottom row
- A roll area greater than the bottom row minus the top row

This API corresponds directly to the 5250 Roll command. See the 5250 data stream documentation for details.

Restrictions

The following considerations apply to the QsnRollDown API:

- Lines vacated due to a roll are not cleared to nulls.
- The command does not change the format table, and so, should be avoided when it could produce discrepancies between the format table and the display.
- Data rolled out of the roll area are lost.

Required Parameter Group

Number of lines to roll
INPUT; BINARY(4)
The number of lines to roll the designated area down by.

Top row of roll area
INPUT; BINARY(4)
The line number defining the top line of the area that will participate in the roll.

Bottom row of roll area
INPUT; BINARY(4)
The line number defining the bottom line of the area that will participate in the roll.

Optional Parameter Group

Command buffer handle
INPUT; BINARY(4)
If this parameter is omitted or specified as 0, this is a direct operation and the screen is rolled down immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA315 E Roll parameters not valid.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Roll Up (QsnRollUp) API

Parameters

Required Parameter Group:

1	Number of lines to roll	Input	Binary(4)
2	Top row of roll area	Input	Binary(4)
3	Bottom row of roll area	Input	Binary(4)

Optional Parameter Group:

4	Command buffer handle	Input	Binary(4)
5	Low-level environment handle	Input	Binary(4)
6	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

Usage Notes

The Roll Up (QsnRollUp) API is identical in function to the QsnRollDown API, except the lines are rolled up instead of down.

Required Parameter Group

Number of lines to roll
INPUT; BINARY(4)
The number of lines to roll the designated area up by.

Top row of roll area
INPUT; BINARY(4)
The line number defining the top line of the area that will participate in the roll.

Bottom row of roll area
INPUT; BINARY(4)
The line number defining the bottom line of the area that will participate in the roll.

Optional Parameter Group

Command buffer handle
INPUT; BINARY(4)
If this parameter is omitted or specified as 0, this is a direct operation and the screen is rolled up immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA315 E Roll parameters not valid.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Save Screen (QsnSavScr) API

Parameters

Optional Parameter Group:

1	Saved data command buffer handle	Output	Binary(4)
2	Command buffer handle	Input	Binary(4)
3	Low-level environment handle	Input	Binary(4)
4	Error code	I/O	Char(*)

Returned Value:

	Saved data command buffer handle	Output	Binary(4)

The Save Screen (QsnSavScr) API saves the current state of the display. If this is a direct operation, the API creates a command buffer that contains the operations used to restore the screen state and returns a handle for this buffer. When a direct save screen operation is issued, the saved screen data is returned in a command buffer. This command buffer contains the Restore Screen command along with the data to restore the screen. Additional commands can be added to the command buffer as described in "Restrictions" on page 15-11 under the Restore Screen (QsnRstScr) API.

The screen can be restored by sending this command buffer using the Put Command Buffer (QsnPutBuf) API.

When an indirect save screen operation is issued, the Save Screen command is stored in the command buffer and is considered an input operation. The command can be issued to the screen only through the Put Command Buffer and Perform Get (QsnPutGetBuf) API. The saved data will be returned in the input buffer parameter specified for the QsnPutGetBuf API. The screen can subsequently be restored by specifying this input buffer on the QsnRstScr API.

This command corresponds directly to the 5250 Save Screen (when the underlying control unit supports it) or Save Partial Screen command. See the 5250 data stream documentation for details.

Restrictions

This command must be the last command in the command buffer, except when GUI support is available. In this case, other input commands may follow.

Optional Parameter Group

Saved data command buffer handle
OUTPUT; BINARY(4)
The variable that will contain the command buffer handle for the restore screen operation if this is a direct operation.

For an indirect operation, the result of the save screen will be returned in the input buffer of a subsequent input operation, which can be used to restore the screen using the QsnRstScr operation.

Command buffer handle
INPUT; BINARY(4)
If this parameter is omitted or specified as 0, this is a direct operation and the screen is saved immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Saved data
OUTPUT; BINARY(4)
For a successful operation, this API returns the value for the saved data parameter if this is a direct operation; otherwise, it returns zero. For an unsuccessful operation, it returns -1.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA314 E Memory allocation error.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Set Low-Level Environment Window Mode (QsnSetEnvWinMod) API

Parameters

Required Parameter:			
1	Enable window mode	Input	Char(1)
Optional Parameter Group:			
2	Previous window mode setting	Output	Char(1)
3	Window mode description	Input	Char(*)
4	Length of window mode description	Input	Binary(4)
5	Low-level environment handle	Input	Binary(4)
6	Error Code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Set Low-Level Environment Window Mode (QsnSetEnvWinMod) API enables or disables the window mode for the low-level environment. Use this API to affect how row and cursor positions specified on low-level interface operations are interpreted for operations to the given low-level environment. Additional details regarding windows can be found in Chapter 19, "Introduction to the Window Services APIs" on page 19-1.

When window mode is enabled, screen locations and cursor positions specified and retrieved in the low-level interface routines are interpreted relative to the logical window area defined. The logical window area is treated as a logical

screen in terms of validity of cursor and data starting positions. If an attempt is made to write data or define a field that starts outside of the current window area, a CPFA31D error is issued for that particular API. However, if data or a field is written to the window that exceeds the window boundary, no error condition occurs and the data or field will extend beyond the window.

The size of the logical window area can be determined through the Retrieve Low-Level Environment Window Mode (QsnRtvEnvWinMod) API. When window mode is enabled, screen addresses must be explicitly specified for APIs such as the QsnWrtDta API in order for the address to be interpreted relative to the window area. If a screen address is not specified, the current display address will be used as an absolute screen address irrespective of the current window mode. This is because the current screen address is not tracked by DSM, but is handled by the control unit.

When window mode is enabled, APIs where a cursor position is specified, such as the QsnSetFld API, issue a CPFA307 error if the given cursor position is outside the bounds of the current window area. For APIs that return a cursor position, such as the Get Cursor Address (QsnGetCsrAdr) API, both the row and column returned will be -1 if the cursor screen location is outside of the current window area, 0 if the cursor is on the top or left border, or the number of screen rows or columns plus 1 if the cursor is on the bottom or right border of the window area, respectively. The following low-level APIs are affected by the window mode:

- QsnRtvReadAdr
- QsnRtvFldInf
- QsnGetCsrAdr
- QsnGetCsrAdrAID
- QsnSetOutAdr
- QsnWrtDta
- QsnWrtSFMaj
- QsnWrtTDta
- QsnWrtExtAtr
- QsnWrtPad
- QsnWrtPadAdr
- QsnSetFld
- QsnSetCsrAdr
- QsnInsCsr
- QsnSetErr

The actual screen location used for a screen I/O operation is calculated using the formula `base+offset=actual`, where `base` is the upper left row/column location of the window border (0 for full screen) if `offset` is positive and the lower right row/column location of the window border (screen height/width plus 1 for full screen), if `offset` is negative, `offset` is the row/column specified on the API, and `actual` is the actual screen location. For example, if the window area were defined to be from row 3, column 10 with 15 rows and 30 columns, as shown in Figure 15-1, then an attempt to position the cursor through the QsnSetCsrAdr API at row 4, column 5 would actually position the cursor on the screen at row 7, column 15, as indicated by the letter `a` in Figure 15-1. Specifying row 9, column -7 would position the cursor on the screen at row 12, column 34 (`b` in Figure 15-1). An attempt to position the cursor at row 16, column 5 would result in a CPFA307 error (Screen position &1,&2 outside of display or window area.), since this position is outside the bounds of the current window area. Given the same window area description, a call to the QsnRtvFldInf API specifying an input buffer containing a field read from row 10, column 20 on the actual screen (the `c` in Figure 15-1), would return a field row and column location of 7 and 10 respectively. A call to the QsnGetCsrAdr API would return -1,-1 if the cursor were located outside of the window area, such as in row 18 column 32.

Enabling or disabling the window mode does not affect any data currently displayed on the screen or the behavior of any commands stored previously in a command buffer. For example, if the window mode were enabled as described above and the QsnSetCsrAdr API was invoked as an indirect operation specifying row 4 and column 5, the cursor position command stored in the command buffer would reflect the current window mode. Whenever that command buffer is written out, the cursor would always be set on the screen at row 6, column 14, regardless of whether or not window mode was disabled or changed at the point when the command buffer was written to the screen.

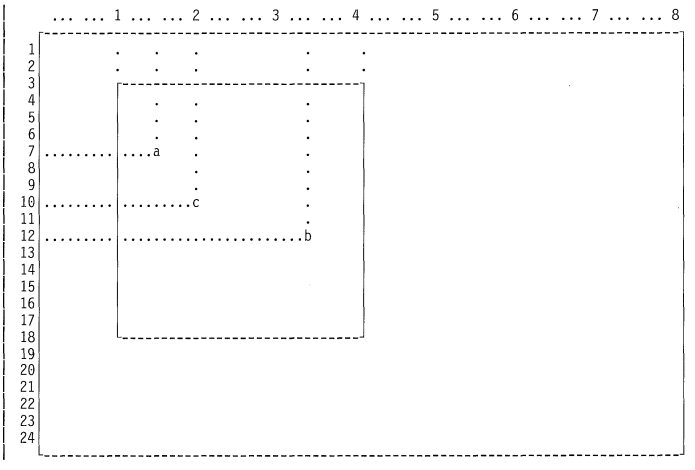


Figure 15-1. Window Area

Required Parameter

Window mode

INPUT; CHAR(1)

Whether window mode should be enabled or disabled. The possible values are:

- **0** Disable window mode
- **1** Enable window mode

Window mode is initially disabled.

Optional Parameter Group

Previous window mode setting
OUTPUT; CHAR(1)
Whether window mode was enabled or disabled prior to this API being called. The possible values returned are:

0 Window mode was disabled prior to this API being called
1 Window mode was enabled prior to this API being called

Window mode description
INPUT; CHAR(*)
The window mode description. If this parameter is omitted or the length parameter is specified as 0, and window mode is to be enabled, the values from the previous window mode setting will be used. If no such values exist, the current screen size will be used. The window area described must fall within the bounds of the current screen size. An CPFA31C will be issued if an invalid window area is specified. The format of this field is shown in "Set Low-Level Environment Window Mode (QsnSetEnvWinMod) API" on page 15-17.

This parameter is ignored if window mode is to be disabled.

Length of window mode description
INPUT; BINARY(4)
The length of the window mode description parameter. If this parameter is specified, it must be either 0, in which case the window mode description parameter is ignored, or exactly 13 bytes in which case the window mode description is required.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Window Mode Description

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Attribute column indication
1	1	BINARY(4)	Upper left row of window area border
5	5	BINARY(4)	Upper left column of window area border
9	9	BINARY(4)	Number of rows in window area
13	D	BINARY(4)	Number of columns in window area

Field Descriptions

Attribute column indication. Whether the column containing the left border of the logical window is an attribute column. Operations such as QsnWrtDta can specify column 1 for the data location and specify a leading attribute. In this case the data will be written to the first column of the window area and the attribute will be written to the column containing the logical window border. If the attribute column is not specified for the window area, such an operation would result in a CPFA31D error (Attempt to write outside of window area.). This column would also be used to insert the leading attribute when line wrapping occurs within the window.

The allowable values are:

0 No attribute column
1 Attributes can be written to column containing left logical window border

Number of columns in window area. The number of columns in the window area.

Number of rows in window area. The number of rows in the window area.

Upper left column of window area border. The column location of the leftmost column in the window area. This parameter must be a value between 0 and the screen width inclusive.

Upper left row of window border. The row location of the upper border of the logical window area. This parameter must be a value between 0 and the screen height inclusive.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.
CPFA324 E Window area definition is incorrect.
CPFA32A E Window mode indication value must be 0 or 1.
CPFA334 E Low-level environment handle incorrect.

Chapter 16. Buffer Manipulation and Query APIs

The buffer manipulation and query APIs are used to create, query, and manipulate input and command buffers that interact with the screen.

The two buffer types used by the low-level interfaces are command and input. A command buffer can be used to accumulate a sequence of low-level commands without performing an I/O operation. The entire buffer can be written out at once in a single I/O operation. See "Direct and Indirect Operations" on page 14-1 for further discussion of command buffers. When an input operation is performed, you can specify an input buffer as the target of the operation. The input results are placed in this buffer, which can then be queried through several interfaces. You may query for individual pieces of information, such as the AID byte from the input operation using the Retrieve AID Code on Read (QsnRtvReadAID) API or for multiple pieces of information using the Retrieve Read Information (QsnRtvReadInf) API.

The buffer manipulation and query interfaces include the following:

- **Clear Buffer** (QsnClrBuf) clears all commands or data in a buffer resets its state.
- **Copy Buffer** (QsnCpyBuf) copies the contents of one buffer to another.
- **Create Command Buffer** (QsnCrtCmdBuf) creates a command buffer to accumulate low-level commands.
- **Create Input Buffer** (QsnCrtInpBuf) creates an input buffer to receive input results.
- **Delete Buffer** (QsnDltBuf) deletes a buffer.
- **Put Command Buffer** (QsnPutBuf) sends the commands in a command buffer to the screen.
- **Put Command Buffer and Perform Get** (QsnPutGetBuf) sends the commands in a command buffer to the screen and performs a read operation.
- **Retrieve AID Code on Read** (QsnRtvReadAID) determines the Aid code for a given input operation.
- **Retrieve Buffer Data Length** (QsnRtvBufLen) returns the length of data in a buffer.
- **Retrieve Buffer Size** (QsnRtvBufSiz) returns the size of a buffer.
- **Retrieve Cursor Address on Read** (QsnRtvReadAdr) retrieves the cursor position at the completion of an input operation.
- **Retrieve Field Information** (QsnRtvFldInf) returns information about a particular field in an input buffer.
- **Retrieve Length of Data in Input Buffer** (QsnRtvDtaLen) retrieves the number of data bytes in an input buffer after an input operation.
- **Retrieve Length of Field Data in Buffer** (QsnRtvFldDtaLen) retrieves the number of bytes of field data after an input operation.
- **Retrieve Number of Bytes Read from Screen** (QsnRtvReadLen) retrieves the number of data bytes read from the screen after an input operation.
- **Retrieve Number of Fields Read** (QsnRtvFldCnt) retrieves the number of fields in an input buffer.
- **Retrieve Pointer to Data in Input Buffer** (QsnRtvDta) returns a pointer to the first byte of input data in an input buffer.
- **Retrieve Pointer to Field Data** (QsnRtvFldDta) returns a pointer to the first byte of field data in an input buffer.
- **Retrieve Read Information** (QsnRtvReadInf) returns information about the input operation.

The detailed API descriptions are presented in alphabetical order.

Clear Buffer (QsnClrBuf) API

Parameters

Required Parameter:			
1	Buffer handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Clear Buffer (QsnClrBuf) API clears all commands or data and resets the state of the given buffer. This is the only API that clears or removes data in a buffer.

Required Parameter

Buffer handle
INPUT; BINARY(4)
A handle for the buffer to be cleared. The storage associated with the buffer is not deallocated.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The

value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.

Copy Buffer (QsnCpyBuf) API

Parameters

Required Parameter Group:			
1	Source buffer handle	Input	Binary(4)
2	Target buffer handle	Input	Binary(4)
Optional Parameter:			
3	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Copy Buffer (QsnCpyBuf) API copies the contents of one buffer to another buffer. Both buffers must be the same type—command or input. If the target and source buffers are the same, no operation takes place and no error is reported.

If a target command buffer contains data, the data in the source buffer is appended to the target buffer. A CPFA301 error is issued if the target command buffer is not large enough to hold the contents of the source buffer and cannot be resized.

If input buffers are being copied, the target buffer must be empty. If the target input buffer is not large enough to hold the data from the source buffer, the data is truncated and no error is reported.

Required Parameter Group

Source buffer handle
INPUT; BINARY(4)
A handle for the buffer from which data is to be copied. The contents of this buffer are not affected by this operation.

Target buffer handle
INPUT; BINARY(4)
A handle for the buffer to which data is to be copied.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA305 E Cannot add operation to command buffer.
CPFA301 E Command buffer is full.
CPFA313 E Command buffer already contains an input operation.
CPFA31E E Required parameter &1 omitted.
CPFA330 E Buffer type mismatch.
CPFA331 E Buffer handle incorrect.

Create Command Buffer (QsnCrtCmdBuf) API

Parameters

Required Parameter:			
1	Initial command buffer size	Input	Binary(4)
Optional Parameter Group:			
2	Increment amount	Input	Binary(4)
3	Maximum size	Input	Binary(4)
4	Command buffer handle	Output	Binary(4)
5	Error code	I/O	Char(*)
Returned Value:			
	Command buffer handle	Output	Binary(4)

The Create Command Buffer (QsnCrtCmdBuf) API creates a command buffer for use with low-level operations that accept a command buffer parameter.

Required Parameter

Initial command buffer size
INPUT; BINARY(4)
The initial size of the command buffer, in bytes, to create. This parameter must be greater than 0 and less than the size of the underlying display file I/O buffer:

approximately 4500 bytes for 24x80, 6300 bytes for 27x132, 8000 bytes for DBCS-capable displays, 8800 bytes for DBCS presentation screen-capable displays, and 16000 bytes for DBCS ideographic-capable displays.

Optional Parameter Group

Increment amount
INPUT; BINARY(4)
The amount to increment the command buffer size by if there is not enough space to store a specified command. If this parameter is omitted or specified with a zero value, the buffer size will not be incremented and a CPFA301 error will be issued when there is no space in the buffer to store a requested command. If an attempt is made to increment a command buffer to a size that exceeds the available memory resources or the size of the underlying display file I/O buffer, the increment will not take place and a CPFA301 error will be issued for that operation.

Maximum size
INPUT; BINARY(4)
The maximum size to increment the command buffer to when there is not enough space to store a specified command. If this parameter is nonzero, it must be greater than the initial command buffer size parameter, and less than the size of the underlying display file I/O buffer. If this parameter is omitted or specified with a zero value, no maximum value is assigned for the command buffer. If the buffer is to be incremented, it will be incremented until either there is no additional storage available or the command buffer exceeds the size of the display file I/O buffer. If the increment amount parameter is omitted or specified with a zero value, this parameter is ignored and the maximum size is the same as the initial command buffer size.

Command buffer handle
OUTPUT; BINARY(4)
The variable containing the handle for the command buffer created after the QsnCrtCmdBuf API has completed. The buffer state will be the same as that following a QsnClrBuf operation.

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.

Returned Value

Command buffer handle
OUTPUT; BINARY(4)
This API returns the value for the command buffer handle parameter if successful, or -1 otherwise.

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.
CPFA312 E Buffer size parameter error.
CPFA314 E Memory allocation error.

Create Input Buffer (QsnCrtInpBuf) API

Parameters

Required Parameter:

1	Input buffer size	Input	Binary(4)

Optional Parameter Group:

2	Increment amount	Input	Binary(4)
3	Maximum size	Input	Binary(4)
4	Input buffer handle	Output	Binary(4)
5	Error code	I/O	Char(*)

Returned Value:

	Input buffer handle	Output	Binary(4)

The Create Input Buffer (QsnCrtInpBuf) API creates an input buffer for use with low-level commands that accept an input buffer parameter.

Required Parameter

Input buffer size
INPUT; BINARY(4)
The size of the input buffer, in bytes, to create. This parameter must be greater than 0 and less than the size of the underlying display file I/O buffer: approximately 4500 bytes for 24x80, 6300 bytes for 27x132, 8000 bytes for DBCS-capable displays, 8800 bytes for DBCS presentation screen-capable displays, and 16000 bytes for DBCS ideographic-capable displays.

Optional Parameter Group

Increment amount
INPUT; BINARY(4)
The amount to increment the buffer size by if there is not enough space to store a read operation. If this parameter is omitted or specified with a zero value, the buffer size is not be incremented and input data is truncated if there is not enough space.

Maximum size
INPUT; BINARY(4)
The maximum size to increment the input buffer to when there is not enough space to store the result of a read operation. If this parameter is nonzero, it must be greater than the initial command buffer size parameter, and less than the size of the underlying display file I/O

buffer. If this parameter is omitted or specified with a zero value, no maximum value is assigned for the command buffer, if the buffer is to be incremented, it will be incremented until either there is no additional storage available or the command buffer exceeds the size of the display file I/O buffer. If the increment amount parameter is omitted or specified with a zero value, this parameter is ignored and the maximum size is the same as the initial command buffer size.

Input buffer handle
OUTPUT; BINARY(4)
The variable containing the handle for the created input buffer after the QsnCrtInpBuf API has completed. The buffer state becomes the same as that following a QsnClrBuf operation.

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.

Returned Value

Input buffer handle
OUTPUT; BINARY(4)
This API returns the value for the input buffer handle parameter, or -1 otherwise.

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.
CPFA312 E Buffer size parameter error.
CPFA314 E Memory allocation error.

Delete Buffer (QsnDltBuf) API

Parameters

Required Parameter:			
1	Buffer handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Delete Buffer (QsnDltBuf) API deletes a command or input buffer created with the Create Command Buffer (QsnCrtCmdBuf) or the Create Input Buffer (QsnCrtInpBuf) API, respectively. All storage associated with the buffer is deallocated.

Required Parameter

Buffer handle
Input; BINARY(4)
A handle for the buffer to be deleted.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.

Put Command Buffer (QsnPutBuf) API

Parameters

Required Parameter:			
1	Command buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	Low-level environment handle	Input	Binary(4)
3	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Put Command Buffer (QsnPutBuf) API sends the commands accumulated in a command buffer to the screen. This corresponds to a write operation to the display file. If the command buffer contains no data, the operation returns successfully, but no I/O operation is performed.

Required Parameter

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer. The command buffer is not modified in any way as a result of this command.

Optional Parameter Group

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA309 E Invalid cursor position in command buffer.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.
CPFA338 E Command buffer contains an input operation.

Put Command Buffer and Perform Get (QsnPutGetBuf) API

Parameters

Required Parameter Group:

1	Command buffer handle	Input	Binary(4)
2	Input buffer handle	Input	Binary(4)

Optional Parameter Group:

3	Low-level environment handle	Input	Binary(4)
4	Error code	I/O	Char(*)

Returned Value:

Return code	Output	Binary(4)

The Put Command Buffer and Perform Get (QsnPutGetBuf) API sends the commands accumulated in a command buffer to the screen and performs a read operation. The command buffer must contain an input operation. If it has no input operation, a CPFA333 error occurs.

Required Parameter Group

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer to be sent to the screen.

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that receives the result of the input operation.

Optional Parameter Group

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
| CPFA303 E Error occurred during screen I/O operation.
| CPFA304 E Data-stream error &1 reported for screen I/O operation.
| CPFA309 E Invalid cursor position in command buffer.
| CPFA31E E Required parameter &1 omitted.
| CPFA326 E Screen must be redrawn.
| CPFA331 E Buffer handle incorrect.
| CPFA334 E Low-level environment handle incorrect.
| CPFA33E E Command buffer does not contain an input operation.

Retrieve AID Code on Read (QsnRtvReadAID) API

Parameters

Required Parameter:			
1	Input buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	AID code	Output	Char(1)
3	Error code	I/O	Char(*)
Returned Value			
	AID code	Output	Char(1)

The Retrieve AID Code on Read (QsnRtvReadAID) API determines the AID code corresponding to the input operation that filled the given input buffer.

Required Parameter

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation. The input buffer must be filled as a result of a Read Input Fields (QsnReadInp), Read Modified Fields (QsnReadMDT), or Read Modified Alternate (QsnReadMDTAlt) operation. If the input buffer is filled as a result of any other input operation, a CPFA32E error is issued.

Optional Parameter Group

AID code
OUTPUT; CHAR(1)
The variable that contains the AID code when the QsnRtvReadAID API has completed. See "AID-Generating Keys" on page 14-2 for a description of the possible values.

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.

Returned Value

AID code
OUTPUT; CHAR(1)
This API returns the value for the AID code parameter or X'00' if an error occurs during processing.

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.
| CPFA319 E No data in input buffer.
| CPFA31E E Required parameter &1 omitted.
| CPFA32E E Input data for query operation incorrect.
| CPFA32F E Buffer type incorrect.
| CPFA331 E Buffer handle incorrect.
| CPFA334 E Low-level environment handle incorrect.

Retrieve Buffer Data Length (QsnRtvBufLen) API

Parameters

Required Parameter:			
1	Buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	Buffer data length	Output	Binary(4)
3	Error code	I/O	Char(*)
Returned Value:			
	Buffer data length	Output	Binary(4)

The Retrieve Buffer Data Length (QsnRtvBufLen) API returns the number of bytes of command data in a command buffer or of input data in an input buffer. After an indirect operation is applied to a command buffer, the QsnRtvBufLen API result reflects the increase in the underlying command stream to accommodate the command.

Required Parameter

Buffer handle
INPUT; BINARY(4)
A handle for the buffer to be queried.

Optional Parameter Group

Buffer data length
OUTPUT; BINARY(4)
The variable containing the buffer data length after the QsnRtvBufLen API has completed.

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.

Returned Value

Buffer data length
OUTPUT; BINARY(4)
This API returns the value for the buffer data length parameter, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.

Retrieve Buffer Size (QsnRtvBufSiz) API

Parameters

Required Parameter:			
1	Buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	Buffer size	Output	Binary(4)
3	Error code	I/O	Char(*)
Returned Value:			
	Buffer size	Output	Binary(4)

The Retrieve Buffer Size (QsnRtvBufSiz) API returns the total number of bytes allocated for a command or input buffer. The result returned from this API is the current allocated buffer size, including any increments that may have taken place.

Required Parameter

Buffer handle
INPUT; BINARY(4)
A handle for the buffer to be queried.

Optional Parameter Group

Buffer size
OUTPUT; BINARY(4)
The variable containing the buffer size after the QsnRtvBufSiz API has completed.

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.

Returned Value

Buffer size
OUTPUT; BINARY(4)
This API returns the value for the buffer size parameter, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.

Retrieve Cursor Address on Read (QsnRtvReadAdr) API

Parameters

Required Parameter:			
1	Input buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	Cursor row	Output	Binary(4)
3	Cursor column	Output	Binary(4)
4	Low-level environment handle	Input	Binary(4)
5	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Retrieve Cursor Address on Read (QsnRtvReadAdr) API determines the row and column position of the cursor when the input operation that filled the given input buffer has completed. You must specify at least one of the cursor row or the cursor column parameter. If both of these parameters are omitted, a CPFA31E error occurs.

The input buffer must be filled as a result of a Read Input Fields (QsnReadInp), Read Modified Fields (QsnReadMDT), Read Modified Alternate (QsnReadMDTAlt), Read Immediate

(QsnReadImm), or Read Modified Immediate Alternate (QsnReadMDTImmAlt) operation. If the input buffer is filled as a result of any other input operation, a CPFA32E message is issued.

Required Parameter

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation.

Optional Parameter Group

Cursor row
OUTPUT; BINARY(4)
The variable that contains the row position of the cursor when the QsnRtvReadAdr API has completed.

Cursor column
OUTPUT; BINARY(4)
The variable that contains the column position of the cursor when the QsnRtvReadAdr API has completed.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA319 E No data in input buffer.
CPFA31E E Required parameter &1 omitted.
CPFA32E E Input data for query operation incorrect.
CPFA32F E Buffer type incorrect.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Retrieve Field Information (QsnRtvFldInf) API

Parameters

Required Parameter Group:

1	Input buffer handle	Input	Binary(4)
2	Field number	Input	Binary(4)
3	Receiver variable	Output	Char(*)
4	Length of receiver variable	Input	Bin(4)

Optional Parameter Group:

5	Low-level environment handle	Input	Binary(4)
6	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Retrieve Field Information (QsnRtvFldInf) API retrieves information about a field in an input buffer filled by a Read Modified Fields (QsnReadMDT), Read Modified Alternate (QsnReadMDTAlt), or Read Modified Immediate Alternate (QsnReadMDTImmAlt) operation.

To query the results from a Read Input Fields (QsnReadInp) or Read Immediate (QsnReadImm) operation, use the Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) and Retrieve Pointer to Field Data (QsnRtvFldDta) APIs. To query the result from any other input operations, use the Retrieve Length of Data in Input Buffer (QsnRtvDtaLen) and Retrieve Pointer to Data in Input Buffer (QsnRtvDta) APIs.

Required Parameter Group

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation.

Field number
INPUT; BINARY(4)
The number of the field to query, specified as n, where n is the nth field in the input buffer.

Receiver variable
Output; CHAR(*)
The structure that will contain the result of the query when the QsnRtvFldInf API has completed.

Length of receiver variable
Input; BINARY(4)
The length of the receiver variable parameter.

Optional Parameter Group

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Query Input Field Result

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(1)	Type of field
9	9	BINARY(4)	Row position of field
13	D	BINARY(4)	Column position of field
17	11	BINARY(4)	Length of data read
21	15	CHAR(11)	Reserved
32	20	PTR(SPP)	Pointer to field data

Field Descriptions

Bytes available. The number of bytes of data available to be returned.

Bytes returned. The number of bytes of data returned.

Column position of field. The column position of the specified field on the screen. If the field type is transparent, this parameter is set to 0.

Length of data read. The length of the data read from the specified field.

Pointer to field data. A pointer to the data for the specified field.

Row position of field. The row position of the specified field on the screen. If the field type is transparent, this parameter is set to 0.

Type of field. The type of the specified field. The possible values are:

Value	Description
1	Normal field
2	Transparent field

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C24 E Length of the receiver variable is not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA319 E No data in input buffer.
CPFA31A E Incorrect field number value &1 specified.
CPFA31E E Required parameter &1 omitted.
CPFA32E E Input data for query operation incorrect.
CPFA32F E Buffer type incorrect.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Retrieve Length of Data in Input Buffer (QsnRtvDtaLen) API

Parameters

Required Parameter:			
1	Input buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	Input data length	Output	Binary(4)
3	Error code	I/O	Char(*)
Returned Value			
	Input data length	Output	Binary(4)

The Retrieve Length of Data in Input Buffer (QsnRtvDtaLen) API determines the number of bytes of input data contained in an input buffer after an input operation.

Required Parameter

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation.

Optional Parameter Group

Input data length
OUTPUT; BINARY(4)
The variable that contains the input data length when the QsnRtvDtaLen API has completed. This number may be smaller than the number of bytes actually read if the input buffer was not large enough to hold all the data.

Use the Retrieve Number of Bytes Read from Screen (QsnRtvReadLen) API to determine the amount of data actually read from the screen. If the value returned by the QsnRtvReadLen API is less than the input data length, then truncation of the input data occurred.

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.

Returned Value

Input data length
OUTPUT; BINARY(4)
This API returns the value for the input data length parameter, or -1 otherwise.

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.
CPFA319 E No data in input buffer.
CPFA31E E Required parameter &1 omitted.
CPFA32F E Buffer type incorrect.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) API

Parameters

Required Parameter:			
1	Input buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	Length of field data	Output	Binary(4)
3	Error code	I/O	Char(*)
Returned Value			
	Length of field data	Output	Binary(4)

The Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) API determines the number of bytes of field data returned after a Read Input Fields (QsnReadInp) or Read Immediate (QsnReadImm) input operation. You can use the Retrieve Pointer to Field Data (QsnRtvFldDta) API to retrieve a pointer to this data so that you can parse the field values. Refer to the Read Input Fields (QsnReadInp) API for a description of the format of the data returned.

To query the results from a Read Modified Fields (QsnReadMDT), Read Modified Alternate (QsnReadMDTAlt), or Read Modified Immediate Alternate (QsnReadMDTImmAlt) operation, use the Retrieve Number of Fields Read (QsnRtvFldCnt) and Retrieve Field Information (QsnRtvFldInf) APIs. To query the result from any other input operation, use the Retrieve Length of Data in Input Buffer (QsnRtvDtaLen) and Retrieve Pointer to Data in Input Buffer (QsnRtvDta) APIs.

Required Parameter

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation. The input buffer must be filled as a result of a QsnReadInp or QsnReadImm operation.

Optional Parameter Group

Length of field data
OUTPUT; BINARY(4)
The variable that contains the field data length when the QsnRtvFldDtaLen API has completed. The field data length is 3 bytes less than the value returned by the QsnRtvDtaLen API. (The cursor and AID-key values account for the first 3 bytes of the input data 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. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Returned Value

Length of field data
OUTPUT; BINARY(4)
This API returns the value for the length of field data parameter, or -1 otherwise.

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.
CPFA319 E No data in input buffer.
CPFA31E E Required parameter &1 omitted.
CPFA32E E Input data for query operation incorrect.
CPFA32F E Buffer type incorrect.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Retrieve Number of Bytes Read from Screen (QsnRtvReadLen) API

Parameters

Required Parameter:			
1	Input buffer handle	Input	Binary(4)
Optional Parameter Group			
2	Read data length	Output	Binary(4)
3	Error code	I/O	Char(*)
Returned Value			
	Read data length	Output	Binary(4)

The Retrieve Number of Bytes Read from Screen (QsnRtvReadLen) API returns the number of bytes of data read from the screen into an input buffer after an input operation.

Required Parameter

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation.

Optional Parameter Group

Read data length
OUTPUT; BINARY(4)
The variable that contains the read data length when the QsnRtvReadLen API has completed. This number may be larger than the number of bytes actually contained in the buffer if the input buffer was not large enough to hold all the data. Use the Retrieve Length of Data in Input Buffer (QsnRtvDtaLen) API to determine the amount of data contained in the buffer or to determine if truncation occurred.

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.

Returned Value

Read data length
OUTPUT; BINARY(4)
This API returns the value for the read data length parameter, or -1 otherwise.

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.
CPFA319 E No data in input buffer.
CPFA31E E Required parameter &1 omitted.
CPFA320 E Pointer parameter is null.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Retrieve Number of Fields Read (QsnRtvFldCnt) API

Parameters

Required Parameter:			
1	Input buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	Field count	Output	Binary(4)
3	Error code	I/O	Char(*)
Returned Value			
	Field count	Output	Binary(4)

The Retrieve Number of Fields Read (QsnRtvFldCnt) API returns the number of fields contained in an input buffer after a Read Modified Fields (QsnReadMDT), Read Modified Alternate (QsnReadMDTAlt), or Read Modified Immediate Alternate (QsnReadMDTImmAlt) operation. Use the Retrieve Field Information (QsnRtvFldInf) API to retrieve information about a specific field.

To query the results from a QsnReadInp or QsnReadImm operation, use the Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) and Retrieve Pointer to Field Data (QsnRtvFldDta) APIs. To query the result from any other input operation, use the Retrieve Length of Data in Input Buffer (QsnRtvDtaLen) and Retrieve Pointer to Data in Input Buffer (QsnRtvDta) APIs.

Required Parameter

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation. The input buffer must be filled as a result of a QsnReadMDT, QsnReadMDTAlt, or QsnReadMDTImmAlt operation.

Optional Parameter Group

Field count
OUTPUT; BINARY(4)
The variable that contains the field count when the QsnRtvFldCnt API has completed.

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.

Returned Value

Field count
OUTPUT; BINARY(4)
This API returns the value for the field count parameter, or -1 otherwise.

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.
CPFA314 E Memory allocation error.
CPFA319 E No data in input buffer.
CPFA31E E Required parameter &1 omitted.
CPFA32E E Input data for query operation incorrect.
CPFA32F E Buffer type incorrect.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Retrieve Pointer to Data in Input Buffer (QsnRtvDta) API

Parameters

Required Parameter:			
1	Input buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	Pointer to input data	Output	PTR(SPP)
3	Error code	I/O	Char(*)
Returned Value			
	Pointer to input data	Output	PTR(SPP)

The Retrieve Pointer to Data in Input Buffer (QsnRtvDta) API returns a pointer to the first byte of input data in an input buffer after a read operation.

Required Parameter

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation.

Optional Parameter Group

Pointer to input data
OUTPUT; PTR(SPP)
The variable that contains the pointer to the input data after the QsnRtvDta API has completed. You can use the Retrieve Length of Data in Input Buffer (QsnRtvDtaLen) API to retrieve the length of this data. Refer to the appropriate read operation for a description of the format of the data returned. The value returned by this API is equivalent to the data returned by the system on an input operation. This parameter must be on a 16-byte boundary.

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.

Returned Value

Pointer to input data
OUTPUT; PTR(SPP)
This API returns the value for the pointer to input data parameter, or the null pointer otherwise.

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.
CPFA319 E No data in input buffer.
CPFA31E E Required parameter &1 omitted.
CPF3C1F E Pointer parameter is not on a 16-byte boundary.
CPFA32F E Buffer type incorrect.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Retrieve Pointer to Field Data (QsnRtvFldDta) API

Parameters

Required Parameter:			
1	Input buffer handle	Input	Binary(4)
Optional Parameter Group:			
2	Pointer to field data	Output	PTR(SPP)
3	Error code	I/O	Char(*)
Returned Value			
	Pointer to field data	Output	PTR(SPP)

The Retrieve Pointer to Field Data (QsnRtvFldDta) API returns a pointer to the first byte of field data in an input buffer after a Read Input Fields (QsnReadInp), Read Immediate (QsnReadImm), Read Modified Fields (QsnReadMDT), Read Modified Alternate (QsnReadMDTAlt), or Read Modified Immediate Alternate (QsnReadMDTImmAlt) operation. You can use the Retrieve Length of Field Data in Buffer

(QsnRtvFldDtaLen) API to retrieve the length of this data. Refer to the "Read Input Fields (QsnReadInp) API" on page 17-5 for a description of the format of the data returned.

To query the results from a QsnReadMDT, QsnReadMDTAlt, or QsnReadMDTImmAlt operation, you can also use the QsnRtvFldCnt and QsnRtvFldInf APIs. To query the result from any other input operations, use the QsnRtvDtaLen and QsnRtvDta APIs.

Required Parameter

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation. The input buffer must be filled as a result of a QsnReadInp or QsnReadImm operation.

Optional Parameter Group

Pointer to field data
OUTPUT; PTR(SPP)
The variable that contains the pointer to the field data when the QsnRtvFldDta API has completed. The value returned by this API is the null pointer if the buffer contains no field data. Otherwise, it is equivalent to adding 3 bytes to the address returned by QsnRtvDta API. (The cursor and AID key values account for the first 3 bytes of input data returned.) This parameter must be on a 16-byte boundary.

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.

Returned Value

Pointer to field data
OUTPUT; PTR(SPP)
This API returns the value for the pointer to field data parameter, or the null pointer otherwise.

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.
CPFA319 E No data in input buffer.
CPFA31E E Required parameter &1 omitted.
CPF3C1F E Pointer parameter is not on a 16-byte boundary.
CPFA32E E Input data for query operation incorrect.
CPFA32F E Buffer type incorrect.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Retrieve Read Information (QsnRtvReadInf) API

Parameters

Required Parameter Group:			
1	Input buffer handle	Input	Binary(4)
2	Query result	Output	Char(*)
3	Length of query result	Input	Binary(4)
Optional Parameter Group:			
4	Low-level environment handle	Input	Binary(4)
5	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Retrieve Read Information (QsnRtvReadInf) API returns information about the input operation that filled the given input buffer.

Required Parameter Group

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that contains the results of the input operation.

Query result
OUTPUT; CHAR(*)
The structure that contains the result of the query when the QsnRtvReadInf API has completed. The format of this structure is shown in "Format of the Query Result" on page 16-14.

Length of query result
INPUT; BINARY(4)
The length of the query result parameter. The minimum value must be 8.

Optional Parameter Group

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Query Result

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(8)	Reserved
16	10	PTR(SPP)	Pointer to first byte of data
32	20	PTR(SPP)	Pointer to first byte of field data
48	30	BINARY(4)	Number of bytes of input data
52	34	BINARY(4)	Number of bytes of field data
56	38	BINARY(4)	Number of fields in input buffer
60	3C	BINARY(4)	Number of bytes of data sent
64	40	BINARY(4)	Row location of cursor
68	44	BINARY(4)	Column location of cursor
72	48	CHAR(1)	AID code for AID-associated read request

Field Descriptions

Bytes available. The number of bytes of data available to be returned.

Bytes returned. The number of bytes of data returned.

AID code for AID-associated read request. AID code corresponding to key pressed to service an AID-associated read request. The input buffer must be filled as a result of a QsnReadInp, QsnReadMDT, or QsnReadMDTAlt operation. If the input buffer is filled as a result of any other input operation, this field is set to X'00'. See "AID-Generating Keys" on page 14-2 for a description of the possible values.

Column location of cursor. Column location of cursor when the input operation was serviced. The input buffer must be filled as a result of a QsnReadInp, QsnReadMDT, QsnReadMDTAlt, QsnReadImm, or QsnReadMDTImmAlt operation. If the input buffer is filled as a result of any other input operation, this field is set to -1.

Number of bytes of data sent. Number of bytes of data sent from screen. If this value is larger than the number of bytes of input data, then truncation occurs on the input operation.

Number of bytes of field data. Number of bytes of field data in input buffer. This does not include the 3 bytes of header information (the cursor row and column, and the AID byte). The input buffer must be filled as a result of a QsnReadInp, QsnReadMDT, QsnReadMDTAlt, QsnReadImm, or QsnReadMDTImmAlt operation. If the input buffer is filled as a result of any other input operation, this field is set to -1.

Number of bytes of input data. Number of bytes of input data in input buffer. This includes header information such as row and column position.

Number of fields in input buffer. Number of fields in input buffer. This does not include header information such as row and column position. The input buffer must be filled as a result of a QsnReadMDT, QsnReadMDTAlt, or QsnReadMDTImmAlt operation. If the input buffer is filled as a result of any other input operation, or the input data format cannot be determined, this field is set to -1.

Pointer to first byte of data. Pointer to first byte of data in input buffer. This includes header information such as row and column position.

Pointer to first byte of field data. Pointer to first byte of field data in input buffer. This will be the first byte of data following the header information (the cursor row and column, and the AID byte). If the buffer does not contain field data, this field is set to the null pointer.

Row location of cursor. Row location of cursor when the input operation was serviced. The input buffer must be filled as a result of a QsnReadInp, QsnReadMDT, QsnReadMDTAlt, QsnReadImm, or QsnReadMDTImmAlt operation. If the input buffer is filled as a result of any other input operation, this field is set to -1.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1F E Pointer parameter is not on a 16-byte boundary.
CPF3C24 E Length of the receiver variable is not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA319 E No data in input buffer.
CPFA31E E Required parameter &1 omitted.
CPFA320 E Pointer parameter is null.
CPFA32F E Buffer type incorrect.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Chapter 17. Screen Input APIs

The screen input APIs allow you to read data and other information from the screen. This includes field data, screen attributes, and the cursor address. The screen-input interfaces correspond, either directly or indirectly, to the 5250 data stream read commands.

The screen input APIs are:

Get AID (QsnGetAID) waits for an AID-generating key to be pressed.
Get Cursor Address (QsnGetCsrAdr) gets the current cursor address.
Get Cursor Address with AID (QsnGetCsrAdrAID) gets the current cursor address after an AID-generating key is pressed.
Put Input Command (QsnPutInpCmd) issues a supplied read command.
Read Immediate (QsnReadImm) reads the contents of all input fields on the display without requiring an AID key to be pressed.
Read Input Fields (QsnReadInp) reads the contents of all input fields on the display requiring an AID key to be pressed.
Read Modified Alternate (QsnReadMDTAlt) reads the contents of all modified fields on the display, alternate form, requiring an AID key to be pressed.
Read Modified Fields (QsnReadMDT) reads the contents of all modified fields requiring an AID key to be pressed.
Read Modified Immediate Alternate (QsnReadMDTImmAlt) reads the contents of all modified fields on the display, alternate form, without requiring an AID key to be pressed.
Read Screen (QsnReadScr) reads the contents of the screen without requiring an AID key to be pressed.

The detailed API descriptions are presented in alphabetical order.

Read Command Processing

The read commands fall into two categories– AID-associated and immediate. When an AID-associated read command is issued, an AID-generating key must be pressed at the terminal for control to return to the program. (The keyboard must be unlocked to allow an AID key to be pressed.) For direct read operations or indirect operations where the associated command buffer does not contain a Write to Display command, DSM implicitly issues a Write to Display (QsnWTD) operation with control characters QSN_CC1_NULL and QSN_CC2_UNLOCKBD before issuing the read command.

At most, one read command can be issued with a given screen I/O operation. An attempt to add a read operation to a command buffer that already has one results in an error. If a command buffer contains a read command, the command buffer must be sent to the screen using the Put Command Buffer and Perform Get (QsnPutGetBuf) API. An attempt to send such a buffer using the Put Command Buffer (QsnPutBuf) API will result in an error.

Get AID (QsnGetAID) API

Parameters

Optional Parameter Group:			
1	AID code	Output	Char(1)
2	Low-level environment handle	Input	Binary(4)
3	Error code	I/O	Char(*)
Returned Value			
	AID code	Output	Char(1)

The Get AID (QsnGetAID) API waits for an AID-generating key to be pressed.

This command corresponds indirectly to the 5250 Read Input Fields command. Because the control characters specified on the underlying command are both X'00', this operation will cause the cursor to move to the insert cursor position when the keyboard is unlocked.

Optional Parameter Group

AID code
OUTPUT; CHAR(1)
The variable that contains the AID code when the QsnRtvReadAID API has completed.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

AID code
OUTPUT; CHAR(1)
This API returns the value for the AID code parameter, or X'00' if an error occurs during processing.

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.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA326 E Screen must be redrawn.
CPFA334 E Low-level environment handle incorrect.

Get Cursor Address (QsnGetCsrAdr) API

Parameters

Optional Parameter Group:

1	Cursor row	Output	Binary(4)
2	Cursor column	Output	Binary(4)
3	Low-level environment handle	Input	Binary(4)
4	Error code	I/O	Char(*)

Returned Value:

Return code	Output	Binary(4)

The Get Cursor Address (QsnGetCsrAdr) API returns the current cursor address, without requiring an AID-generating key to be pressed. Either the cursor row or cursor column parameter must be specified. If one of these parameters is omitted, a CPFA31E error occurs.

This command corresponds indirectly to the 5250 Read Immediate command.

Restrictions

The same restrictions apply as for the "Read Immediate (QsnReadImm) API" on page 17-4.

Optional Parameter Group

Cursor row
OUTPUT; BINARY(4)
The variable that contains the cursor row when the QsnGetCsrAdr API has completed.

Cursor column
OUTPUT; BINARY(4)
The variable that contains the cursor column when the QsnGetCsrAdr API has completed.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.
CPFA334 E Low-level environment handle incorrect.

Get Cursor Address with AID (QsnGetCsrAdrAID) API

Parameters

Optional Parameter Group:

1	Cursor row	Output	Binary(4)
2	Cursor column	Output	Binary(4)
3	AID code	Output	Char(1)
4	Low-level environment handle	Input	Binary(4)
5	Error code	I/O	Char(*)

Returned Value:

Return code	Output	Binary(4)

The Get Cursor Address with AID (QsnGetCsrAdrAID) API returns the cursor address after an AID-generating key is pressed. Either the cursor row or the cursor column parameter must be specified. If both of these parameters are omitted, a CPFA31E error occurs.

This command corresponds indirectly to the 5250 Read Input Fields command. Because the control characters specified on the underlying command are both X'00', this operation may cause the cursor to move to the default, or insert cursor, position when the keyboard is unlocked.

Optional Parameter Group

Cursor row
OUTPUT; BINARY(4)
The variable that contains the cursor row when the QsnGetCsrAdrAID API has completed.

Cursor column
OUTPUT; BINARY(4)
The variable that contains the cursor column when the QsnGetCsrAdrAID API has completed.

AID code
OUTPUT; CHAR(1)
The variable that contains the AID code when the QsnGetCsrAdrAID API has completed.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA31E E Required parameter &1 omitted.
CPFA326 E Screen must be redrawn.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Put Input Command (QsnPutInpCmd) API

Parameters

Required Parameter:

1	Command	Input	Char(1)

Optional Parameter Group:

2	Command data	Input	Char(*)
3	Command data length	Input	Binary(4)
4	Number of data bytes read	Output	Binary(4)
5	Input buffer handle	Input	Binary(4)
6	Command buffer handle	Input	Binary(4)
7	Low-level environment handle	Input	Binary(4)
8	Error code	I/O	Char(*)

Returned Value:

	Number of field data bytes read	Output	Binary(4)

The Put Input Command (QsnPutInpCmd) API is used to issue data stream input commands that are not directly supported through a DSM API. An Escape (X'04') character is inserted in the stream directly before the command itself for both direct and indirect operations.

You must use this operation to issue an input command that is not directly supported by DSM as this will cause the appropriate underlying screen I/O operation to occur in order to retrieve input. You cannot, for example, use the Put Output Command (QsnPutOutCmd) API to issue an input command because no input data will be requested by the underlying DSM screen I/O operation. For example, to issue the display data stream command `Read Screen with Extended Attributes`, specify X'64' for the command parameter.

The command buffer handle or input buffer handle parameter must be specified as follows:

- The command buffer handle is specified with a nonzero value. The input buffer handle is omitted or specified with a zero value.

  This is an indirect operation. The command is stored in the command buffer without an I/O operation taking place.

- The command buffer handle is omitted or specified with a zero value. The input buffer handle is specified with a nonzero value.

  This is a direct operation. The input operation is issued to the screen, and the resulting input data is stored in the input buffer.

- Both a command buffer handle and an input buffer handle are specified with nonzero values.

  This is a direct operation. The input operation is appended to the command stream given by the command buffer, and the entire command stream is written to the display. The resulting input data is stored in the input buffer. The contents of the command buffer

are not affected by this operation. That is, the input operation is not stored in the command buffer.

This operation corresponds to an Escape character followed by the specified command.

Required Parameter

Command
INPUT; CHAR(1)
The 1-byte character code for the input command to be issued. For example, to issue a Save Partial Screen command, the command data should contain X'03' and the command data will contain the dimensions of the partial screen to be saved.

Optional Parameter Group

Command data
INPUT; CHAR(*)
The data for the command to be issued.

Command data length
INPUT; BINARY(4)
The length of the command data parameter.

Number of data bytes read
OUTPUT; BINARY(4)
The variable that contains the number of data bytes returned after the QsnPutInpCmd API has completed if a direct operation is specified. The parameter is not modified for an indirect operation and the value remains unchanged from whatever was passed.

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that receives the result of the input operation if a direct operation is specified.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Number of data bytes read
OUTPUT; BINARY(4)
This API returns the value for the number of data bytes read parameter if a direct operation was specified, or -1 if an error occurs during processing. If this is an indirect operation, this API returns zero if successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA313 E Command buffer already contains an input operation.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA333 E Parameter &1 not positive integer value.
CPFA334 E Low-level environment handle incorrect.

Read Immediate (QsnReadImm) API

Parameters

Optional Parameter Group:

1	Number of field data bytes read	Output	Binary(4)
2	Input buffer handle	Input	Binary(4)
3	Command buffer handle	Input	Binary(4)
4	Low-level environment handle	Input	Binary(4)
5	Error code	I/O	Char(*)

Returned Value:

Number of field data bytes read	Output	Binary(4)

The Read Immediate (QsnReadImm) API reads the contents of all input fields on the display without requiring an AID key to be pressed. The command buffer handle or input buffer handle parameter must be specified as described in "Put Input Command (QsnPutInpCmd) API" on page 17-3.

The information returned depends on the condition of the master MDT bit. (See "Modified Data Tag (MDT) Bit" on page 14-5.) If the bit is not set, the input returned consists of the cursor address and an AID code only. If the bit is set, the input returned also includes the field data in the data portion of the input buffer. In each case, the returned cursor address indicates the current location of the cursor and an AID code of X'00'. The format of the field data returned is

the same as that for the Read Input Fields (QsnReadInp) API.

This command corresponds directly to the 5250 Read Immediate command.

Restrictions

This command must be the last command in the command buffer. A CPFA305 error is issued if there is a subsequent attempt to add another command to the specified command buffer after this command.

Optional Parameter Group

Number of field data bytes read
OUTPUT; BINARY(4)
The variable that contains the number of field data bytes returned after the QsnReadImm API has completed if a direct operation is specified. The parameter is not modified for an indirect operation and the value remains unchanged from whatever was passed.

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that receives the result of the input operation if a direct operation is specified. The result can be queried using the input buffer query operations. See "Retrieve Pointer to Field Data (QsnRtvFldDta) API" on page 16-12 and "Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) API" on page 16-10 .

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Number of field data bytes read
OUTPUT; BINARY(4)
This API returns the value for the number of field data bytes read parameter if a direct operation was specified, or -1 if an error occurs during processing. If this is an indirect operation, this API returns zero if successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA302 E Command buffer or input buffer parameters required.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA309 E Invalid cursor position in command buffer.
CPFA313 E Command buffer already contains an input operation.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Read Input Fields (QsnReadInp) API

Parameters

Required Parameter Group:

1	Control character byte 1	Input	Char(1)
2	Control character byte 2	Input	Char(1)

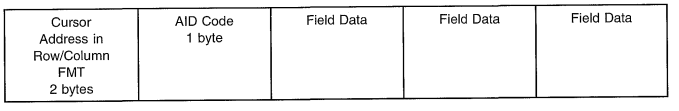
Optional Parameter Group:

3	Number of field data bytes read	Output	Binary(4)
4	Input buffer handle	Input	Binary(4)
5	Command buffer handle	Input	Binary(4)
6	Low-level environment handle	Input	Binary(4)
7	Error code	I/O	Char(*)

Returned Value:

	Number of field data bytes read	Output	Binary(4)

The Read Input Fields (QsnReadInp) API reads the contents of all input fields on the display that require an AID-generating key to be pressed. The command buffer handle or input buffer handle parameter must be specified as described in "Put Input Command (QsnPutInpCmd) API" on page 17-3. The format of the data returned is:



The information returned depends on the condition of the master MDT bit (see "Modified Data Tag (MDT) Bit" on page 14-5) and the AID-generating key pressed. If the bit is not set, the input returned consists of the cursor address and an AID code only. If the bit is set, the input returned also includes the field data in the data portion of the input buffer. In either case, the returned cursor address indicates the

location of the cursor when the AID-generating key was pressed and the AID code for the AID-generating key the operator used. See "AID-Generating Keys" on page 14-2 for a description of the AID-generating character values.

Field data is returned only when one of the following AID-generating keys is used:

- Roll Up
- Roll Down
- Enter/Auto Record Advance
- Auto Enter
- An unmasked command function key

The field data, when returned, consists of the contents of all input fields as they appear on the display, unless resequencing has been specified. (See "Resequencing" on page 14-5.) Any attributes contained in a field are treated as data and returned as such. The attributes that start and end a field are not returned. These are considered to be outside the boundaries of the field. No field delimiters are added; data from each field is followed directly by the data from the next field. Data for nontransparent fields is formatted as follows:

- All nulls (leading, embedded, or trailing) are converted to blanks.
- If the specified field is a signed numeric field:
  - The last character of the field is stripped off.
  - The last location of the field is checked for a negative sign. If detected, the zone portion of the second to last character of the field is changed to a X'D'.

The data returned for transparent fields is not edited. A transparent field is defined with a transparency field control word X'84yy'. (See "Set Field (QsnSetFld) API" on page 18-10.) Each field read is returned unedited with no intervening control information. If a field is both transparent and signed numeric, unpredictable results can occur in the field data.

This command corresponds directly to the 5250 Read Input Fields command.

Restrictions

This command cannot be issued if the control unit supports ideographic data types. A CPFA306 error will occur if an attempt is made to issue this command to a control unit that supports ideographic data types.

Some control units, like those emulated by the PC Support/400 program, do not support a control character associated with input commands. For such units, the control character specified would be ignored. A program could cause further actions to be suspended if, for example, the control character byte 2 specified to unlock the keyboard and this action was not specified elsewhere in the data stream. If the underlying control unit does not support a control character with input commands, you must specify the action to perform using the Write to Display (QsnWTD) API.

Required Parameter Group

Control character byte 1
INPUT; CHAR(1)
The operation for the display to perform after the read operation has been serviced. See "Control Characters" on page 14-2 for a description of the control character values.

Control character byte 2
INPUT; CHAR(1)
The operation for the display to perform after the read operation and control character byte 1 have been serviced. See "Control Characters" on page 14-2 for a description of the control character values.

Optional Parameter Group

Number of field data bytes read
OUTPUT; BINARY(4)
The variable that contains the number of field data bytes returned after the QsnReadInp API has completed if a direct operation is specified. The parameter is not modified for an indirect operation and the value remains unchanged from whatever was passed.

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that receives the result of the input operation if a direct operation is specified. The result can be queried using the input buffer query operations. See "Retrieve Pointer to Field Data (QsnRtvFldDta) API" on page 16-12 and "Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) API" on page 16-10.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Number of field data bytes read
OUTPUT; BINARY(4)
This API returns the value for the number of field data bytes read parameter if a direct operation was specified, or -1 if an error occurs during processing. If this is an indirect operation, this API returns zero if successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA302 E Command buffer or input buffer parameters required.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA309 E Invalid cursor position in command buffer.
CPFA313 E Command buffer already contains an input operation.
CPFA31C E Incorrect value for control character byte &1.
CPFA31E E Required parameter &1 omitted.
CPFA326 E Screen must be redrawn.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Read Modified Alternate (QsnReadMDTAlt) API

Parameters

Required Parameter Group:

1	Control character byte 1	Input	Char(1)
2	Control character byte 1	Input	Char(1)

Optional Parameter Group:

3	Field count	Output	Binary(4)
4	Input buffer handle	Input	Binary(4)
5	Command buffer handle	Input	Binary(4)
6	Low-level environment handle	Input	Binary(4)
7	Error code	I/O	Char(*)

Returned Value:

	Field count	Output	Binary(4)

The Read Modified Alternate (QsnReadMDTAlt) API reads the contents of all modified fields on the screen, alternate form, requiring an AID-generating key to be pressed. The QsnReadMDTAlt API is functionally equivalent to the QsnReadMDT API with the following exceptions:

- Leading and embedded nulls within the fields remain nulls. However, trailing nulls are stripped off.
- For fields consisting entirely of nulls, but with their MDT bit on, only the field's address is returned.

See "Read Modified Fields (QsnReadMDT) API" on page 17-8 for details.

This command corresponds directly to the 5250 Read MDT Alternate command.

Restrictions

This command is not supported by all control units. A CPFA306 error occurs if an attempt is made to issue this command to a control unit that does not support it.

Some control units, like those emulated by the PC Support/400 program, do not support a control character associated with input commands. For such units, the control character specified would be ignored. A program could cause further actions to be suspended if, for example, the control character byte 2 specified to unlock the keyboard and this action was not specified elsewhere in the data stream. If the underlying control unit does not support a control character with input commands, you must specify the action to perform using the Write to Display (QsnWTD) API.

Required Parameter Group

Control character byte 1
INPUT; CHAR(1)
The operation for the display to perform after the read operation has been serviced. See "Control Characters" on page 14-2 for a description of the control character values.

Control character byte 2
INPUT; CHAR(1)
The operation for the display to perform after the read operation and control character byte 1 have been serviced. See "Control Characters" on page 14-2 for a description of the control character values.

Optional Parameter Group

Field count
OUTPUT; BINARY(4)
The variable that will contain the number of input fields read after the QsnReadMDTAlt API has completed, if a direct operation is specified. The parameter is not modified for an indirect operation and the value remains unchanged from whatever was passed.

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that receives the result of the input operation if a direct operation is specified. The result can be queried using the input buffer query operations. See "Retrieve Pointer to Field Data (QsnRtvFldDta) API" on page 16-12 and "Retrieve

Length of Field Data in Buffer (QsnRtvFldDtaLen) API" on page 16-10.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Field count
OUTPUT; BINARY(4)
Returns the value for the field count parameter if a direct operation was specified or -1 if an error occurs during processing. If this is an indirect operation, returns zero if successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA302 E Command buffer or input buffer parameters required.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA306 E Command not supported by current device.
CPFA309 E Invalid cursor position in command buffer.
CPFA313 E Command buffer already contains an input operation.
CPFA31C E Incorrect value for control character byte &1.
CPFA31E E Required parameter &1 omitted.
CPFA326 E Screen must be redrawn.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Read Modified Fields (QsnReadMDT) API

Parameters

Required Parameter Group:

1	Control character byte 1	Input	Char(1)
2	Control character byte 2	Input	Char(1)

Optional Parameter Group:

3	Field count	Output	Binary(4)
4	Input buffer handle	Input	Binary(4)
5	Command buffer handle	Input	Binary(4)
6	Low-level environment handle	Input	Binary(4)
7	Error code	I/O	Char(*)

Returned Value:

	Field count	Output	Binary(4)

The Read Modified Fields (QsnReadMDT) API reads the contents of all modified fields on the screen requiring an AID-generating key to be pressed. The command buffer handle or input buffer handle parameter must be specified as described in "Put Input Command (QsnPutInpCmd) API" on page 17-3. See "Control Characters" on page 14-2 for a description of the control character values. The format of the data returned is:

Cursor Row/ Column 2 bytes	AID Code 1 byte	SBA X'11'	Field Row/ Column 2 bytes	Field Data	SBA X'11'	Field Row/ Column 2 bytes	Field Data

The information returned depends on the state of the MDT bit for each field (see "Modified Data Tag (MDT) Bit" on page 14-5) and the AID-generating key used. If no bits are set, the input returned consists of the cursor address and an AID code only. If at least one bit is set, the input may also include field data. In each case, the returned cursor address indicates the location of the cursor when the AID-generating key was pressed and the AID code for the AID-generating key the operator used.

Field data is returned only when one of the following AID-generating keys is used:

Roll Up
Roll Down
Enter/Auto Record Advance
An unmasked command function key

The field data, when returned, consists of the row and column address and the contents of each field that has an MDT bit on as they appear on the display, unless resequencing has been specified. (See "Resequencing" on page 14-5.) The input buffer query routines "Retrieve Number of Fields Read (QsnRtvFldCnt) API" on page 16-11 and "Retrieve Field Information (QsnRtvFldInf) API" on page 16-8 can be used to retrieve the value for each field. (To interpret the data directly, QsnRtvDta can be used to obtain a pointer for the data portion of the input buffer. The

first data byte will be the SBA order for the first field as defined in the 5250 data stream documentation.)

Data for nontransparent fields is formatted as follows:

- Leading and embedded nulls within the field are translated to blanks, and trailing nulls are stripped off. Only the field's address is returned for fields whose MDT bit is on but which consist entirely of nulls.
- If the specified field is a signed numeric field:
  - The last character of the field is stripped off unless it was previously stripped because it was null.
  - The last location of the field is checked for a negative sign. If detected, the zone portion of the second to last character of the field is changed to a X'D'.
- If the field is a transparent field, no formatting is performed.

If a field is both transparent and signed numeric, unpredictable results can occur in the field data.

This command corresponds directly to the 5250 Read MDT Fields command.

Restrictions

Some control units, like those emulated by the PC Support/400 program, do not support a control character associated with input commands. For such units, the control character specified would be ignored. A program could cause further actions to be suspended if, for example, the control character byte 2 specified to unlock the keyboard and this action was not specified elsewhere in the data stream. If the underlying control unit does not support a control character with input commands, you must specify the action to perform using the Write to Display (QsnWTD) API.

Required Parameter Group

Control character byte 1
INPUT; CHAR(1)
The operation for the display to perform after the read operation has been serviced. See "Control Characters" on page 14-2 or a description of the control character values.

Control character byte 2
INPUT; CHAR(1)
The operation for the display to perform after the read operation and control character byte 1 have been serviced. See "Control Characters" on page 14-2 for a description of the control character values.

Optional Parameter Group

Field count
OUTPUT; BINARY(4)
The variable that contains the number of input fields read after the QsnReadMDT API has completed if a direct operation is specified. The parameter is not modified for an indirect operation and the value remains unchanged from whatever was passed.

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that receives the result of the input operation if a direct operation is specified. The result can be queried using the input buffer query operations. See "Retrieve Pointer to Field Data (QsnRtvFldDta) API" on page 16-12 and "Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) API" on page 16-10.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Field count
OUTPUT; BINARY(4)
This API returns the value for the field count parameter if a direct operation was specified, or -1 if an error occurs during processing. If this is an indirect operation, this API returns zero if successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA302 E Command buffer or input buffer parameters required.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA309 E Invalid cursor position in command buffer.
CPFA313 E Command buffer already contains an input operation.
CPFA31C E Incorrect value for control character byte &1.
CPFA31E E Required parameter &1 omitted.
CPFA326 E Screen must be redrawn.
CPFA331 E Buffer handle incorrect.

CPFA334 E Low-level environment handle incorrect.

Read Modified Immediate Alternate (QsnReadMDTImmAlt) API

Parameters

Optional Parameter Group:

1	Field count	Output	Binary(4)
2	Input buffer handle	Input	Binary(4)
3	Command buffer handle	Input	Binary(4)
4	Low-level environment handle	Input	Binary(4)
5	Error code	I/O	Char(*)

Returned Value:

	Field count	Output	Binary(4)

The Read Modified Immediate Alternate (QsnReadMDTImmAlt) API reads the contents of all modified fields on the display without requiring an AID-generating key to be pressed. Processing for this API is the same as for the Read Immediate (QsnReadImm) API, except that data is returned only for those fields that have the MDT bit on. The format of the data returned is the same as for the Read Modified Alternate (QsnReadMDTAlt) API.

See "Read Immediate (QsnReadImm) API" on page 17-4 and "Read Modified Alternate (QsnReadMDTAlt) API" on page 17-7 for details.

This command corresponds directly to the 5250 Read MDT Immediate Alternate command.

Restrictions

This command is not supported by all control units. A CPFA306 error occurs if an attempt is made to issue this command to a control unit that does not support it.

Optional Parameter Group

Field count
OUTPUT; BINARY(4)
The variable that contains the number of input fields read after the QsnReadMDTImmAlt API has completed if a direct operation is specified. The parameter is not modified for an indirect operation and the value remains unchanged from whatever was passed.

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that receives the result of the input operation if a direct operation is specified. The result can be queried using the input buffer query operations. See "Retrieve Pointer to Field Data (QsnRtvFldDta) API" on page 16-12 and "Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) API" on page 16-10.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Field count
OUTPUT; BINARY(4)
This API returns the value for the field count parameter if a direct operation was specified, or -1 if an error occurs during processing. If this is an indirect operation, this API returns zero if successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA302 E Command buffer or input buffer parameters required.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA306 E Command not supported by current device.
CPFA309 E Invalid cursor position in command buffer.
CPFA313 E Command buffer already contains an input operation.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Read Screen (QsnReadScr) API

Parameters

Optional Parameter Group:

1	Number of data bytes read	Output	Binary(4)
2	Input buffer handle	Input	Binary(4)
3	Command buffer handle	Input	Binary(4)
4	Low-level environment handle	Input	Binary(4)
5	Error code	I/O	Char(*)

Returned Value:

	Number of data bytes read	Output	Binary(4)

The Read Screen (QsnReadScr) API reads the contents of the entire screen without requiring an AID-generating key to be pressed. The command buffer handle or input buffer handle parameter must be specified as described in "Put Input Command (QsnPutInpCmd) API" on page 17-3.

The data returned consists of the contents of the entire display, including the attributes. No formatting or conversion is done. The data will be available in the data portion of the input buffer. The result of this operation can be queried using the "Retrieve Length of Data in Input Buffer (QsnRtvDtaLen) API" on page 16-9 and the "Retrieve Pointer to Data in Input Buffer (QsnRtvDta) API" on page 16-12.

This command corresponds directly to the 5250 Read Screen command.

Restrictions

The same restrictions apply as for the "Read Immediate (QsnReadImm) API" on page 17-4. In addition, this command cannot be issued if the control unit supports ideographic data types. A CPFA306 error occurs if an attempt is made to issue this command to a control unit that supports ideographic data types.

Optional Parameter Group

Number of data bytes read
OUTPUT; BINARY(4)
The variable that contains the number of data bytes returned after the QsnReadScr API has completed if a direct operation is specified. The parameter is not modified for an indirect operation and the value remains unchanged from whatever was passed.

Input buffer handle
INPUT; BINARY(4)
A handle for the input buffer that receives the result of the input operation if a direct operation is specified. The result can be queried using the input buffer query operations. See "Retrieve Pointer to Field Data (QsnRtvFldDta) API" on page 16-12 and "Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) API" on page 16-10.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Number of data bytes read
OUTPUT; BINARY(4)
This API returns the value for the number of data bytes read parameter if a direct operation was specified, or -1 if an error occurs during processing. If this is an indirect operation, this API returns zero if successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA302 E Command buffer or input buffer parameters required.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA309 E Invalid cursor position in command buffer.
CPFA313 E Command buffer already contains an input operation.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Chapter 18. Screen Output APIs

The screen output APIs are used to define fields and write data and other information to the screen.

The screen output interfaces comprise the following:

Delete Field ID Definition (QsnDltFldId) deletes a field ID definition.
Generate a Beep (QsnBeep) generates a beep.
Insert Cursor (QsnInsCsr) sets the insert cursor address.
Pad between Two Screen Addresses (QsnWrtPadAdr) pads the screen with characters between two points.
Pad for N Positions (QsnWrtPad) pads the screen for a specified number of characters.
Put Output Command (QsnPutOutCmd) writes a data stream command.
Set Cursor Address (QsnSetCsrAdr) sets the position of the cursor on the screen.
Set Error State (QsnSetErr) places the keyboard into prehelp error state and optionally places a string on the error line with cursor positioning support.
Set Field (QsnSetFld) defines an input field on the screen at a given row and column.
Set Output Address (QsnSetOutAdr) sets the current display address.
Write Data (QsnWrtDta) writes data to the display at a given row and column with standard attributes.
Write to Display (QsnWTD) issues a Write to Display command.
Write Structured Field Major (QsnWrtSFMaj) writes the major structure of a structured field.
Write Structured Field Minor (QsnWrtSFMin) writes the minor structure of a structured field.
Write Transparent Data (QsnWrtTDta) writes transparent data to the display at a given row and column.

The detailed API descriptions are presented in alphabetical order.

Delete Field ID Definition (QsnDltFldId) API

Parameters

Required Parameter:			
1	Field ID	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Delete Field ID Definition (QsnDltFldId) API deletes a field ID definition. The screen appearance, including the fields defined on the screen, are not affected by this command.

Required Parameter

Field ID
INPUT; BINARY(4)
The ID for the field definition to be deleted. Subsequent references to this field ID result in a CPFA33C error. This parameter must be specified with a nonzero valid field value.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA33C E Undefined field ID &1.

Generate a Beep (QsnBeep) API

Parameters

Optional Parameter Group:			
1	Command buffer handle	Input	Binary(4)
2	Low-level environment handle	Input	Binary(4)
3	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Generate a Beep (QsnBeep) API generates a beep. The display address is not affected by this command.

This command corresponds directly to the 5250 Write to Display (WTD) command with control character 1 equal to X'00' and control character 2 equal to X'04'. If this is an indirect operation, this API issues a new WTD command to the command buffer.

Restrictions

The same restrictions apply as for the Write Data (QsnWrtDta) API (see "Restrictions" on page 18-17).

Optional Parameter Group

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the beep is generated immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Insert Cursor (QsnInsCsr) API

Parameters

Optional Parameter Group:

1	Field ID	Input	Binary(4)
2	Cursor row	Input	Binary(4)
3	Cursor column	Input	Binary(4)
4	Command buffer handle	Input	Binary(4)
5	Low-level environment handle	Input	Binary(4)
6	Error code	I/O	Char(*)

Returned Value:

Return code	Output	Binary(4)

The Insert Cursor (QsnInsCsr) API sets the insert-cursor address. The insert-cursor position specifies the home-cursor address. (The position of the cursor when the host system unlocks the keyboard and the display station operator presses the Home key.) The display address is not affected by this command.

If bit 1 of the associated Write to Display command is set to 0 (which is the default), the cursor will be moved on the screen when the QsnInsCsr API is called. To prevent the cursor from being moved, the control character byte 2 bit should be set to 1 and the Write to Display (QsnWTD) operation should be explicitly issued to a command buffer used by the QsnInsCsr API.

This command corresponds indirectly to the 5250 Write to Display (WTD) command with an Insert Cursor order. (For an indirect operation, a WTD is placed in the command buffer only if one does not already exist in that buffer.)

Restrictions

The same restrictions apply as for the Write Data (QsnWrtDta) API.

Optional Parameter Group

Field ID
INPUT; BINARY(4)
The field ID indicating the field at which to set the display address. If this parameter is specified with a nonzero value, the row and column parameters are ignored and the row and column values corresponding to the field ID are used to set the display address. Either the field ID or the row and column parameters must be specified.

Cursor row
INPUT; BINARY(4)
The row at which to position the insert cursor. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula

base+offset=actual. The base is the row location of the top window border (0 for full screen) if offset is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if offset is negative. The offset is the row parameter value specified, and actual is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

Cursor column
INPUT; BINARY(4)
The column at which to position the insert cursor. The column parameter must refer to a column no greater than the current screen or window mode width (if window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula base+offset=actual. The base is the column location of the left window border (0 for full screen) if offset is positive, or the column location of the right window border (screen width plus 1 for full screen) if offset is negative. The offset is the column parameter value specified, and actual is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the insert cursor is positioned at the specified location immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.
CPFA33C E Undefined field ID &1.

Pad between Two Screen Addresses (QsnWrtPadAdr) API

Parameters

Required Parameter Group:

1	Pad character	Input	Char(1)
2	To row	Input	Binary(4)
3	To column	Input	Binary(4)

Optional Parameter Group:

4	From row	Input	Binary(4)
5	From column	Input	Binary(4)
6	Command buffer handle	Input	Binary(4)
7	Low-level environment handle	Input	Binary(4)
8	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Pad between Two Screen Addresses (QsnWrtPadAdr) API pads the display repeatedly with a selected character between two positions on the screen. The current display address is set to the position given by the to-row and to-column values plus one. Padding may occur outside the logical window area defined by the low-level environment window mode setting.

This command corresponds indirectly to the 5250 Write to Display (WTD) command with a Set Buffer Address order (if the from row and from column parameters are specified) and a Repeat to Address order. (For an indirect operation, a WTD is placed in the command buffer only if one does not already exist in that buffer.)

Restrictions

The same restrictions apply as for the Write Data (QsnWrtDta) API.

Required Parameter Group

Pad character
INPUT; CHAR(1)
The character to pad the screen with.

To row
INPUT; BINARY(4)
The row at which to write the last pad character. If the position to pad to is less than the position to pad from, a CPFA31B error is issued. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the row location of the top window border (0 for full screen) if `offset` is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if `offset` is negative. The `offset` is the row parameter value specified, and `actual` is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

To column
INPUT; BINARY(4)
The column at which to write the last pad character. The column parameter must refer to a column no greater than the current screen or window mode width (if window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the column location of the left window border (0 for full screen) if `offset` is positive, or the column location of the right window border (screen width plus 1 for full screen) if `offset` is negative. The `offset` is the column parameter value specified, and `actual` is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Optional Parameter Group

From row
INPUT; BINARY(4)
The row at which to write the first pad character. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the row location of the top window border (0 for full screen) if `offset` is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if `offset` is negative. The `offset` is the row parameter value specified, and `actual` is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

If both the from-row and from-column parameters are omitted, the pad characters are written starting at the current display address. If the command is a direct operation or the buffer specified does not contain a preceding output operation that sets the display address, the current display address is set to row 1, column 1, prior to writing the pad characters. Both the from-row and from-column parameters must be specified, or both parameters must be omitted.

From column
INPUT; BINARY(4)
The column at which to write the first pad character. The column parameter must refer to a column no greater than the current screen or window mode width (if window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the column location of the left window border (0 for full screen) if `offset` is positive, or the column location of the right window border (screen width plus 1 for full screen) if `offset` is negative. The `offset` is the column parameter value specified, and `actual` is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the screen is padded with the character specified between the positions specified inclusively. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA304 E Data-stream error &1 reported for screen I/O operation.

CPFA305 E Cannot add operation to command buffer.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA308 E Attempt to write data past end of display.
CPFA31B E From position &1 greater than to position &2.
CPFA31D E Attempt to write outside of window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.
CPFA335 E Screen address parameter error.
CPFA33C E Undefined field ID &1.

Pad for N Positions (QsnWrtPad) API

Parameters

Required Parameter Group:			
1	Pad character	Input	Char(1)
2	Number of bytes	Input	Binary(4)
Optional Parameter Group:			
3	Field ID	Input	Binary(4)
4	From row	Input	Binary(4)
5	From column	Input	Binary(4)
6	Command buffer handle	Input	Binary(4)
7	Low-level environment handle	Input	Binary(4)
8	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Pad for N Positions (QsnWrtPad) API pads the display with the given pad character for the specified number of bytes. Padding starts at the row and column specified, or at the current display address if these parameters are omitted. To allow the QsnWrtPad operation to insert a group of characters without overwriting the ending screen attribute, the following conditions must be satisfied:

- The operation is an indirect operation.
- The row and column parameters are omitted.
- The low-level environment description does not indicate that DBCS data is being used.
- The previous command saved in the command buffer was a Write Data (QsnWrtDta) operation.

If the row and column parameters are omitted and the command is either a direct operation or the buffer specified does not contain a preceding output operation that sets the display address, then the current display address is set to row 1, column 1, prior to writing the pad characters.

If the from-row and from-column parameters are specified, this command corresponds indirectly to the 5250 Write to Display (WTD) command with a Set Buffer Address order. (For an indirect operation, a WTD is placed in the command buffer only if one does not already exist in the buffer.)

Restrictions

The same restrictions apply as for the Write Data (QsnWrtDta) API.

Required Parameter Group

Pad character
INPUT; CHAR(1)
The pad character to pad the screen with.

Number of bytes
INPUT; BINARY(4)
The number of bytes to pad the screen for.

Optional Parameter Group

Field ID
INPUT; BINARY(4)
The field ID indicating the field at which to set the display address. If this parameter is specified with a nonzero value, the row and column parameters are ignored and the row and column values corresponding to the field ID are used to set the display address. If neither the field ID or row and column parameters are specified, the current display address is used.

From row
INPUT; BINARY(4)
The row at which to write the first pad character. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the row location of the top window border (0 for full screen) if `offset` is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if `offset` is negative. The `offset` is the row parameter value specified, and `actual` is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

If both the from-row and from-column parameters are omitted, the pad characters are written starting at the current display address. If this is the case and the command is a direct operation, or the buffer specified does not contain a preceding output operation that sets the display address, the current display address is set to row 1, column 1, prior to writing the pad characters. Both the from-row and from-column parameters must be specified, or both parameters must be omitted.

From column
INPUT; BINARY(4)
The column at which to write the first pad character. The column parameter must refer to a column no greater than the current screen or window mode width (if

window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the column location of the left window border (0 for full screen) if `offset` is positive, or the column location of the right window border (screen width plus 1 for full screen) if `offset` is negative. The `offset` is the column parameter value specified, and `actual` is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the pad characters are written to the screen at the current display address. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA308 E Attempt to write data past end of display.
CPFA31D E Attempt to write outside of window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA333 E Parameter &1 not positive integer value.
CPFA334 E Low-level environment handle incorrect.
CPFA335 E Screen address parameter error.
CPFA33C E Undefined field ID &1.

Put Output Command (QsnPutOutCmd) API

Parameters

Required Parameter:			
1	Command	Input	Char(1)
Optional Parameter Group:			
2	Command data	Input	Char(*)
3	Command Data Length	Input	Binary(4)
4	Command buffer handle	Input	Binary(4)
5	Low-level environment handle	Input	Binary(4)
6	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Put Output Command (QsnPutOutCmd) API is used to issue data stream commands that are not directly supported through a DSM API. An escape (X'04') character is inserted in the stream directly before the command itself for both direct and indirect operations.

Note: The Write Data (QsnWrtDta) API should be used for issuing Write to Display command orders such as the Write Extended Attributes order.

This operation corresponds to an escape character followed by the specified command.

Required Parameter

Command
INPUT; CHAR(1)
The 1-byte character code for the output command to be issued. For example, to issue a Restore Partial Screen, the command data should contain X'13', the command data will contain the restore data length followed by the restore data, and the command data length will be 2 plus the restore data length.

Optional Parameter Group

Command data
INPUT; CHAR(*)
The data for the command to be issued.

Command data length
INPUT; BINARY(4)
The length of the command data parameter.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the command is sent to the display. If the command being sent is an input command, you must specify a command buffer and then use the Put Command Buffer and Perform Get (QsnPutGetBuf) API to issue the command and retrieve the resulting input. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA333 E Parameter &1 not positive integer value.
CPFA334 E Low-level environment handle incorrect.

Set Cursor Address (QsnSetCsrAdr) API

Parameters

Optional Parameter Group:

1	Field ID	Input	Binary(4)
2	Cursor row	Input	Binary(4)
3	Cursor column	Input	Binary(4)
4	Command buffer handle	Input	Binary(4)
5	Low-level environment handle	Input	Binary(4)
6	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Set Cursor Address (QsnSetCsrAdr) API sets the position of the cursor on the screen. The display address is not affected by this command.

This command corresponds indirectly to the 5250 Write to Display command (WTD) with an Insert Cursor or Move Cursor order. (For an indirect operation, a WTD is placed in the command buffer only if one does not already exist in that buffer.) The Move Cursor order is used if the control unit supports it (based on the 5250 Query command). Otherwise, the Insert Cursor order is used.

If the Move Cursor order is supported, this API sets the cursor without modifying the home address and without regard to the state of the keyboard. Otherwise, the behavior is the same as that of the Insert Cursor (QsnInsCsr) API.

If multiple QsnSetCsrAdr operations are applied to the same command buffer, only the last QsnSetCsrAdr operation is in effect. The last QsnSetCsrAdr or QsnInsCsr operation determines the cursor position. If the Move Cursor order is used, the QsnSetCsrAdr negates any previous QsnInsCsr commands in the command buffer, except for the last QsnInsCsr, which sets the home position. If the Insert Cursor order is used, it negates any previous QsnInsCsr operations. If the Move Cursor order is supported, you can set the home position and then move the cursor by issuing a QsnInsCsr first, followed by a QsnSetCsrAdr operation.

Restrictions

The same restrictions apply as for the Write Data (QsnWrtDta) API.

Optional Parameter Group

Field ID
INPUT; BINARY(4)
The field ID indicating the field at which to set the display address. If this parameter is specified with a nonzero value, the row and column parameters are ignored and the row and column values corresponding to the field ID are used to set the display address. Either

the field ID or the row and column parameters must be specified.

Cursor row
INPUT; BINARY(4)
The row at which to position the cursor. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the row location of the top window border (0 for full screen) if `offset` is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if `offset` is negative. The `offset` is the row parameter value specified, and `actual` is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

Cursor column
INPUT; BINARY(4)
The column at which to position the cursor. The column parameter must refer to a column no greater than the current screen or window mode width (if window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the column location of the left window border (0 for full screen) if `offset` is positive, or the column location of the right window border (screen width plus 1 for full screen) if `offset` is negative. The `offset` is the column parameter value specified, and `actual` is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the cursor is positioned at the specified location immediately. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.
CPFA33C E Undefined field ID &1.

Set Error State (QsnSetErr) API

Parameters

Optional Parameter Group:

1	Message	Input	Char(*)
2	Message length	Input	Binary(4)
3	Field ID	Input	Binary(4)
4	Cursor row	Input	Binary(4)
5	Cursor column	Input	Binary(4)
6	Starting Monochrome attribute	Input	Char(1)
7	Ending Monochrome attribute	Input	Char(1)
8	Starting Color attribute	Input	Char(1)
9	Ending Color attribute	Input	Char(1)
10	Command buffer handle	Input	Binary(4)
11	Low-level environment handle	Input	Binary(4)
12	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Set Error State (QsnSetErr) API places the keyboard into prehelp error state and optionally places a string on the error line.

Either the cursor or the message parameters must be specified to make the command valid. If neither of these are used, a CPFA30F error is issued. If a cursor position is specified, the cursor is moved immediately to the location given. This does not affect the cursor address set by the Insert Cursor (QsnInsCsr) API.

When the operator presses the Help key (prehelp error state only) in response to the error condition, the message `No help text is available` is displayed.

This command corresponds directly to the 5250 Write Error Code command.

Optional Parameter Group

Message
INPUT; CHAR(*)
The message to be displayed. This parameter is required if the message length parameter is specified as a nonzero value. The message data, including the screen attributes, must not exceed 132 characters for devices that are in 27x132 mode, or 80 characters for all other devices. A CPFA310 error is issued if the message data is too long. The message will be truncated to fit.

Message length
INPUT; BINARY(4)
The number of bytes of message data to be displayed.

Field ID
INPUT; BINARY(4)
The field ID indicating the field at which to set the display address. If this parameter is specified with a nonzero value, the row and column parameters are ignored and the row and column values corresponding to the field ID are used to set the display address.

Cursor Row
INPUT; BINARY(4)
The row at which to position the cursor when the message is displayed. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the row location of the top window border (0 for full screen) if `offset` is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if `offset` is negative. The `offset` is the row parameter value specified, and `actual` is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

If the field ID or row and column parameters are omitted, the cursor is not moved. The row and column parameters must be specified together, or both parameters must be omitted.

Cursor Column
INPUT; BINARY(4)
The column at which to position the cursor when the message is displayed. The column parameter must refer to a column no greater than the current screen or window mode width (if window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the column location of the left window border (0 for full screen) if `offset` is positive, or the column location of the right window border (screen width plus 1 for full screen) if `offset` is negative. The `offset` is the column parameter value specified, and `actual` is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Starting monochrome attribute
INPUT; CHAR(1)
The initial screen attribute for monochrome displays. If this parameter is omitted or specified as X'00', a starting attribute of high intensity blink is inserted. See "Screen Attribute Characters" on page 14-3 for a description of the screen attribute values. The starting attribute is selected as for the QsnWrtDta API

Ending monochrome attribute
INPUT; CHAR(1)
The ending screen attribute for monochrome displays. If this parameter is omitted or specified as X'00', an ending attribute of nondisplay is inserted. The ending attribute is selected as for the QsnWrtDta API.

Starting color attribute
INPUT; CHAR(1)
The initial screen attribute for color displays. If this parameter is omitted and color attributes are to be used, no initial attribute is written to the display for the data.

Ending color attribute
INPUT; CHAR(1)
The ending screen attribute for color displays. If this parameter is omitted and color attributes are to be used, no ending attribute is written to the display for the data.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the error state is entered, the cursor is moved to the specified position, and the message, if specified, is displayed. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA30D E Invalid screen attribute.
CPFA30F E Required parameter not specified.
CPFA310 E Error message data/screen attributes exceed display width.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA333 E Parameter &1 not positive integer value.
CPFA334 E Low-level environment handle incorrect.
CPFA335 E Screen address parameter error.
CPFA33C E Undefined field ID &1.
CPFA33F E Error occurred during data conversion.

Set Field (QsnSetFld) API

Parameters

Optional Parameter Group:

1	Field ID	Input	Binary(4)
2	Field length	Input	Binary(4)
3	Row	Input	Binary(4)
4	Column	Input	Binary(4)
5	Field format word (FFW)	Input	Char(2)
6	Field control words (FCW)	Input	Char(*)
7	Number of field control words	Input	Binary(4)
8	Monochrome attribute	Input	Char(1)
9	Color attribute	Input	Char(1)
10	Command buffer handle	Input	Binary(4)
11	Low-level environment handle	Input	Binary(4)
12	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Set Field (QsnSetFld) API defines an input field on the screen at the given row and column. The following occurs when this command is issued to the control unit as a direct operation or when the buffer containing the command is written out:

- Any outstanding AID requests are cleared.
- The keyboard is locked.
- If there is an entry in the format table whose starting address is equal to the address for this field, then that entry is modified. The FFW of the existing entry is replaced by the new FFW and the previous screen starting attribute is overlaid with the new screen starting attribute. The ending screen attribute is not rewritten. All FCWs and the length parameter are ignored. See the 5250 data stream documentation for details.
- If no entry can be found in the table for the field being defined, a new entry will be added to the end of the table. However, the address must be greater than the ending address of the field currently defined last in the format table or an error will occur. If the new entry is valid, it will contain the field's FFW, the optional FCWs, and the field's starting and ending address. An error will occur if an attempt is made to define too many fields on the screen (see the 5250 data stream documentation for details).

The display address after this operation will be the starting field address minus 1 if row and column are specified as valid positive integers or if this is the first field specified within the current WTD command. Otherwise, the display address will be one position past the ending screen attribute.

This command corresponds indirectly to the 5250 Write to Display (WTD) command with a Set Buffer Address order and a Start of Field order if the row and column parameters are specified. (For an indirect operation, a WTD is placed in the command buffer only if one does not already exist in that buffer.)

Restrictions

The same restrictions apply as for the Write Data (QsnWrtDta) API, with the exception that the trailing field attribute can be written past the end of the screen. (It will be suppressed by the control unit.)

Optional Parameter Group

Field ID
INPUT; BINARY(4)
The field ID to be associated with this field. The value specified can be any nonzero integer value. For APIs that accept a field ID parameter, this value can be subsequently used instead of the row and column parameter to specify a screen address. If the given ID is already defined, this operation will redefine that field ID with the values specified. To remove a field ID definition, use the QsnDltFldId API.

If a previously defined field ID is supplied and some or all of the parameters are omitted, the field is defined using the current field definition values for those omitted parameters.

If this field is omitted or specified with a value of zero, then no field ID is associated with this field description.

Field length
INPUT; BINARY(4)
The length of the field being defined. If no field ID is specified, the length must be a positive integer value greater than 1 for signed numeric fields and greater than 0 for all other field types. The entire field must fit on the display. If a field ID is specified with a nonzero value, the length may be 0, in which case a field will not be defined on the screen; however, this will associate the field definition with the specified field ID.

Row
INPUT; BINARY(4)
The row at which to define the field. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the row location of the top window border (0 for full screen) if `offset` is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if `offset` is negative. The `offset` is the row parameter value specified, and `actual` is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

The starting field address will be the row and column locations given if both parameters are specified. Otherwise, it will be the current display address plus 1. If this is the case and the command is a direct operation, or the buffer specified does not contain a preceding output operation that sets the display address, the current display address is set to row 1, column 1, prior to writing the initial screen attribute and the field definition. The ending field address for this field is the starting field address plus the field length.

If a field ID is supplied along with a row and column, the row and column parameters will be stored as specified. These parameters will be used as relative or actual screen positions on a subsequent operation, depending upon the window mode setting for the environment supplied with that operation.

If a previously undefined field ID is supplied with this operation, the row and column parameters must be specified. Also, the row and column parameters must both be specified or omitted; one cannot be specified if the other is omitted. A CPFA307 error occurs if an incorrect cursor position is specified. On some devices, row and column can both be specified as 1, which will cause the field to be defined at row 1, column 1, with a screen attribute of normal (X'20'). If this is the case, then any initial screen attribute parameters specified are ignored. This is only supported by certain devices. Whether or not this is supported can be determined by the Query 5250 (QsnQry5250) API.

Column
INPUT; BINARY(4)
The column at which to write the data. The column parameter must refer to a column no greater than the current screen or window mode width (if window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the column location of the left window border (0 for full screen) if `offset` is positive, or the column location of the right window border (screen width plus 1 for full screen) if `offset` is negative. The `offset` is the column parameter value specified, and `actual` is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Field format word (FFW)
INPUT; CHAR(2)
The field format word is a 2-byte value that controls the type of the field being defined. Figure 18-1 on page 18-12 shows the field types, and the corresponding bit to be set for each type. To omit this parameter, specify X'00' in both characters of the parameter. You must specify this parameter to define an input field, and it is required if a field control word is specified.

Field control words (FCW)
INPUT; CHAR(*)
An array of 2-byte field control words. The field control words are 2-byte values that request certain functions to be performed. Figure 18-2 on page 18-15 shows the valid field control word values, their function, and mnemonics for those values. FCWs will not be checked to see if they are formatted correctly or to see if the function requested is valid for the current device. Errors of this nature may be detected and reported when the FCW is required during subsequent command and keystroke processing. See the 5250 data stream documentation for further details about the meaning and use of these functions.

Number of field control words
INPUT; BINARY(4)
The number of control words in the field control word array. Omitting this parameter or specifying it with a value of 0 indicates that no field control words are specified with the FCW parameter. If this parameter is specified with a nonzero value, the FCW parameter is required; if the FCW parameter is omitted, a CPFA31E error is issued.

Monochrome attribute
INPUT; CHAR(1)
The initial screen attribute for monochrome displays. A screen attribute is required for defining a field on the screen; if this parameter is omitted and monochrome attributes are to be used, X'20' is assumed. The initial screen attribute is written one position to the left of the starting field address. The ending screen attribute

(X'20') is supplied by the controller and written at the end-of-field address plus 1.

The monochrome attribute and color attribute parameters consist of 1 byte that will be used as the screen attribute for a monochrome or a color display, respectively. One of these parameters will be selected based on the underlying display type, and the other will be discarded. See "Screen Attribute Characters" on page 14-3 for a description of the screen attribute values.

Color attribute
INPUT; CHAR(1)
The initial screen attribute for color displays. A screen attribute is required for defining a field on the screen; if this parameter is omitted and color attributes are to be used, X'20' is assumed. See "Screen Attribute Characters" on page 14-3 for a description of the screen attribute values.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the starting field address will be the supplied location. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Field Format Word

The following table shows the field types and the bits to set for each type for the field format word (FFW).

Figure 18-1 (Page 1 of 4). Field Format Words

Field Type	Bit To Set	Mnemonic	Description
	0-15	QSN_NO_FFW	If bits 0 to 15 are set to 0, then no FFW is used. The field will be defined as an output field and its contents will not be returned by operations such as the Read Input Fields (QsnReadInp) API.
	0-1		The first two bits of a field format word must be 01. The mnemonic for every other field type includes this setting.
Bypass	2	QSN_FFW_BYPASS	If this bit is set, this is a bypass field and entries are not allowed in it. If the operator tries to enter something in this field, an error results.
Dup Enable	3	QSN_FFW_DUP	If this bit is set, duplication is allowed in the field. The controller repeats X'1C' from the cursor position to the end of the field when the operator presses the Dup key; this shows on the display as an overstruck asterisk.
Modified data tag (MDT)	4	QSN_FFW_MDT	Setting this bit sets the modified data tag for this field. The field will then be read with the Read Modified Fields (QsnReadMDT) API (see "Read Modified Fields (QsnReadMDT) API" on page 17-8) as if the operator had modified it.
Field Shift/ Edit Specification	**Bits 5-7**	**Mnemonic**	**Description**
Alphabetic shift	000	QSN_FFW_ALPHA_SHIFT	The field accepts all characters. The shift keys are acknowledged. The characters on the lower symbol of each key are valid.

Figure 18-1 (Page 2 of 4). Field Format Words

Alphabetic only	001	QSN_FFW_ALPHA_ONLY	The field accepts only characters A-Z (both uppercase and lowercase) plus the comma (,), period (.), minus (-), space, and DUP (if the DUP-enable bit is on in the associated Field Format Word (FFW)). Other characters cause operator errors. Some special characters for world trade countries are also acceptable (see the 5250 data stream documentation).
Numeric shift	010	QSN_FFW_NUM_SHIFT	The field accepts all characters from all keys.
Numeric only	011	QSN_FFW_NUM_ONLY	The field accepts only characters 0-9 and the comma (,), period (.), minus (-), plus (+), space, and DUP (if the DUP-enable bit is on in the associated Field Format Word (FFW)). Other characters cause operator errors. The unit position of this field will carry the sign digit for the field. If the field is exited with the Field - key, the last character in the field will be 'D' zoned, unless the last character in the field is a '+', '-', ',', '.', or space, in which case an error will be posted. In a right-adjusted field, the field will be right-adjusted before any 'D' zoning or testing of the sign character is performed. When a negative field (from the Field - key) is returned, the units digit will have a 'D' zone.
Katakana shift	100	QSN_FFW_KATA	This is the same as the alphabetic shift except that the keyboard is placed in the Katakana shift on the Japan Katakana data entry, typewriter, and G keyboards. This reverses the order of the cursor direction with respect to the screen. If the display is in bidirectional mode, this changes the cursor direction to left to right; otherwise, it changes the cursor direction to right to left.
Digits only	101	QSN_FFW_DIGIT_ONLY	The field allows keys 0-9 and DUP (if the DUP-enable bit is on in the associated Field Format Word (FFW)).
I/O	110	QSN_FFW_IO	This field will not accept any data keys from the keyboard. An operator error is posted if keystrokes are entered in this field. The operator may move the cursor into and out of this field similar to operation in any non-bypass input field (that is, Field Advance will position the cursor to the start of the field). This field can be used for input from feature devices such as a magnetic stripe reader of selector light pen while data input from the keyboard is excluded. The Field +, Field Exit, and Dup keys are valid for this field and performance is the same as that for any non-bypass input field.

Figure 18-1 (Page 3 of 4). Field Format Words

Signed Numeric	111	QSN_FFW_SIGNED_NUMERIC	The field allows keys 0-9 and DUP (if the DUP-enable bit is on in the associated Field Format Word (FFW)). Typing any other character will cause an operator error display. This field reserves the right-hand position for a sign display (- for negative and null for positive); therefore, the largest number of characters that can be entered into this field is one less than the field length. A signed numeric field less than 2 characters long will cause an error to be flagged. No digit may be keyed into the rightmost position; however, the cursor can be positioned there by using the cursor movement keys and then followed by the F+ or F- key. This allows changing the sign without affecting the rest of the field. If this field is not a mandatory fill or right-adjust field, it is still handled as if it were specified as a right-adjust blank fill field. If the Field - key is used to exit this field, the field will be right-adjusted and a negative sign placed in the rightmost position of the field. The Field Exit or Field + key will insert a blank in the rightmost position and right-adjust this field. Before this field is returned on an input operation, it is changed as follows: • If the rightmost character is a negative sign, the zone of the low order digit is set to a X'D'. • If the rightmost character is not a negative sign, the low order digit is not changed. In either case, the rightmost sign position is not sent to the host.
Field Type	**Bit to Set**	**Mnemonic**	**Description**
Auto Enter	8	QSN_FFW_AUTO_ENTER	If this bit is set, this is an auto enter field. The auto enter function occurs only for valid Field Exit/Field +, Field -, and Dup exit keys, or if the last data character position is typed into the auto enter field. After any required right-adjust, mandatory fill, data duplicating, or check digit operations are successfully performed, the auto enter function causes the panel to be sent to the host as if the Enter key had been pressed.
Field Exit Required	9	QSN_FFW_AUTO_FER	If this bit is set, a nondata key must be typed to leave the field. When the last data character is typed into the last position of the field, the cursor will remain under the character and blink, signifying the controller is waiting for an exit key. Any nondata key will satisfy the exit requirement (including cursor movement or function keys).
Monocase	10	QSN_FFW_AUTO_MONOCASE	If this bit is set, then regardless of the shift state of the typewriter keyboard, only the uppercase A-Z is entered into the field being typed (that is. if a lowercase a is typed, the uppercase A is entered). All other characters are unaffected. Certain characters on some world trade typewriter keyboards will also be translated to uppercase (see the 5250 data stream documentation).
Reserved	11		
Mandatory Enter	12	QSN_FFW_ME	If this bit is set, the operator must enter something in the field before the controller allows the Enter key to be active. The controller recognizes the state of these fields by checking the MDT bit for the field. If the operator tries to bypass the field using a Field +, Field -, or Field Exit key, an error occurs.
Right-Adjust/ Mandatory Fill	**Bits 13-15**	**Mnemonic**	**Description**
No adjust specified	000	QSN_FFW_NOADJUST	No field adjustment occurs.
Reserved	001		
Reserved	010		

Figure 18-1 (Page 4 of 4). Field Format Words

Reserved	011		
Reserved	100		
Right-adjust, zero fill	101	QSN_FFW_RA_ZERO	All leftmost unoccupied positions of a field are filled with zero. Characters are right-adjusted and spaces are zero-filled. The fill character will appear on the display. Right-adjust is only activated by keying the Field Exit, Field +, or Field - keys. The field is right-adjusted from the first non-null character to the left of the cursor when one of these keys is depressed. If a right-adjust field is left through cursor movement keys, the field will remain as is (not right-adjusted). Right-adjust fields longer than 15 characters might cause a slow response that would result in a keyboard overrun. A right-adjust specified field has an implied field exit required function. The Dup key will fill a right-adjust field from the cursor to the end of the field with the Dup character (X'1C'), but the field will not be right-adjusted. After typing the first character into a right adjust field, but prior to exiting the field (using cursor movement or exit keys), the Enter key will cause an operator error to be posted; that is, Enter is invalid from an active right-adjust field.
Right-adjust, blank fill	110	QSN_FFW_RA_BLANK	The field behavior is the same as for right-adjust, zero fill, but the fill character is blank instead of zero.
Mandatory fill	111	QSN_FFW_MF	Once any data has been entered into the field, the field must be completely filled before exiting it. Any attempt to leave an unfilled field causes an error. The cursor may pass into and out of a mandatory fill field as a result of cursor movement keys without fill-checking or error posting by the controller, as long as no data is entered into the field. The Dup key will fill the field from the cursor to the end of field with the Dup characters X'2C', and then the entire field will be checked for any nulls (an error is posted if a null is found). A mandatory fill field with nulls can be returned under the following conditions: 1. The field was initialized with nulls and with the MDT bit on. 2. The Erase Input, Field Exit, or Field + key is used from the first position of the field. The field is filled with nulls and the MDT bit is set on. The above fields, with no further entry, can be returned with all data as blanks on a Read Input Fields (QsnReadInp) operation or as a null field on a Read Modified Fields (QsnReadMDT) operation.

Format of the Field Control Word

Figure 18-2 (Page 1 of 2). Field Control Words

FCW Value	Mnemonic	Description
X'80nn'	QSN_FCW_RESEQ	Entry field resequencing (used in the Read Input Fields (QsnReadInp) and Read Modified Fields (QsnReadMDT) APIs, and so forth). The *nn* specifies the next entry field in the sequence (*nn* can be X'00' to X'80'). See "Resequencing" on page 14-5.
X'8101'	QSN_FCW_MSR	Magnetic stripe reader entry field.
X'8102'	QSN_FCW_SLP	Selector light pen entry field.
X'8103'	QSN_FCW_MSR_SLP	Magnetic stripe reader and selector light pen entry field.
X'8200'	QSN_FCW_DBCS_ONLY	DBCS Only entry field.
X'8220'	QSN_FCW_DBCS_PURE	DBCS Graphic DBCS entry field.

Figure 18-2 (Page 2 of 2). Field Control Words

FCW Value	Mnemonic	Description
X'8240'	QSN_FCW_DBCS_EITHER	DBCS Either entry field.
X'8280' or X'82C0'	QSN_FCW_DBCS_OPEN	DBCS Open entry field.
X'84??'	QSN_FCW_TRANSPARENT	Transparency entry field (used in the Read Input Fields (QsnReadInp) and Read Modified Fields (QsnReadMDT) APIs, and so forth). The *??* indicates that these values are ignored.
X'8501'	QSN_FCW_FET	Forward edge trigger entry field. This provides the same function as Auto Enter specified in the FFW, except a unique AID is returned to the host when the field is exited. The state on the Auto Enter flag in the FFW is ignored.
X'8601'	QSN_FCW_CONT_FIRST	Continued entry field first segment.
X'8603'	QSN_FCW_CONT_LAST	Continued entry field last segment.
X'8602'	QSN_FCW_CONT_MIDDLE	Continued entry field middle segment.
X'88nn'	QSN_FCW_CP	Cursor progression entry field.
X'89nn'	QSN_FCW_HL	Highlighted entry field.
X'8Ann'	QSN_FCW_PDS	Pointer device selection entry field.
X'B140'	QSN_FCW_MOD11	Self Check Modulus 11 entry field.
X'B1A0'	QSN_FCW_MOD10	Self Check Modulus 10 entry field.

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.
CPFA301 E Command buffer is full.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA308 E Attempt to write data past end of display.
CPFA30A E Field length &1 not valid.
CPFA30B E Invalid starting address for field.
CPFA30C E Maximum allowable number of fields exceeded.
CPFA30D E Invalid screen attribute.
CPFA30E E Invalid field format word.
CPFA314 E Memory allocation error.
CPFA31D E Attempt to write outside of window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA332 E Incorrect field control word.
CPFA333 E Parameter &1 not positive integer value.
CPFA334 E Low-level environment handle incorrect.
CPFA335 E Screen address parameter error.
CPFA33D E Field ID definition value &1 not a positive integer.

Set Output Address (QsnSetOutAdr) API

Parameters

Optional Parameter Group:

1	Field ID	Input	Binary(4)
2	Row	Input	Binary(4)
3	Column	Input	Binary(4)
4	Command buffer handle	Input	Binary(4)
5	Low-level environment handle	Input	Binary(4)
6	Error code	I/O	Char(*)

Returned Value:

Return code	Output	Binary(4)

The Set Output Address (QsnSetOutAdr) API sets the current display address. Subsequent output operations that do not reset the display address use this address. If multiple Set Output Address (QsnSetOutAdr) operations are applied to the same command buffer, only the last QsnSetOutAdr operation is in effect.

This command corresponds indirectly to the 5250 Write to Display command (WTD) with a Set Buffer Address order. (For an indirect operation, a WTD is placed in the command buffer only if one does not already exist in that buffer.)

Optional Parameter Group

Field ID
INPUT; BINARY(4)
The field ID indicating the field at which to set the display address. If this parameter is specified with a nonzero value, the row and column parameters are ignored and the row and column values corresponding to

the field ID are used to set the display address. Either the field ID or the row and column parameters must be specified.

Row
INPUT; BINARY(4)
The row at which to set the display address. This parameter is required if the field ID is not specified.

Column
INPUT; BINARY(4)
The column at which to set the display address. This parameter is required if the field ID is not specified.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the display address is set to the specified location. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.
CPFA335 E Screen address parameter error.
CPFA33C E Undefined field ID &1.

Write Data (QsnWrtDta) API

Parameters

Required Parameter Group:

1	Data	Input	Char(*)
2	Data length	Input	Binary(4)

Optional Parameter Group:

3	Field ID	Input	Binary(4)
4	Row	Input	Binary(4)
5	Column	Input	Binary(4)
6	Starting monochrome attribute	Input	Char(1)
7	Ending monochrome attribute	Input	Char(1)
8	Starting color attribute	Input	Char(1)
9	Ending color attribute	Input	Char(1)
10	Command buffer handle	Input	Binary(4)
11	Low-level environment handle	Input	Binary(4)
12	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Write Data (QsnWrtDta) API writes data to the display at a given row and column using standard attributes. If a command buffer is specified that does not contain a previous or current WTD command, one is implicitly added to the buffer using the control characters QSN_CC1_NULL and QSN_CC2_UNLOCKBD. The display address after this operation will be one position past the last data byte written to the screen (including the ending screen attribute, if any).

This command corresponds indirectly to the 5250 Write to Display command (WTD) with a Set Buffer Address order if the row and column are specified. (For an indirect operation, a WTD is placed in the command buffer only if one does not already exist in that buffer.)

Restrictions

If window mode is not set and the data (including attributes) exceeds the length of a row, the data will be wrapped to the next line. If bottom-to-top wrapping is not supported, a CPFA308 error occurs when the data (including attributes) exceeds the last row on the screen. If window mode is set, data that exceeds the width of the window will not be truncated or wrapped within the window, but will wrap across screen rows.

Required Parameter Group

Data
INPUT; CHAR(*)
The data to be written to the screen. If the data being passed is graphic DBCS, the data must be enclosed by extended ideographic attributes. (Use the Write Data (QsnWrtDta) API to specify the data stream Write Extended Attribute order. See the *5250 Functions Reference* for further details.)

Data length
INPUT; BINARY(4)
The number of characters contained in the data parameter.

Optional Parameter Group

Field ID
INPUT; BINARY(4)
The field ID indicating the field at which to set the display address. If this parameter is specified with a nonzero value, the row and column parameters are ignored and the row and column values corresponding to the field ID are used to set the display address. If neither the field ID nor the row and column parameters are specified, the current display address is used.

Row
INPUT; BINARY(4)
The row at which to write the data. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the row location of the top window border (0 for full screen) if `offset` is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if `offset` is negative. The `offset` is the row parameter value specified, and `actual` is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

If both row and column are omitted, the data is written starting at the current display address. If this is the case and the command is a direct operation, or the buffer specified does not contain a preceding output operation that sets the display address, the current display address is set to row 1, column 1 prior to writing the data.

Row and column must both be specified or omitted; one cannot be specified if the other is omitted.

Column
INPUT; BINARY(4)
The column at which to write the data. The column parameter must refer to a column no greater than the current screen or window mode width (if window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the column location of the left window border (0 for full screen) if `offset` is positive, or the column location of the right window border (screen width plus 1 for full screen) if `offset` is negative. The `offset` is the column parameter value specified, and `actual` is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Starting monochrome attribute
INPUT; CHAR(1)
The initial screen attribute for monochrome displays. If this parameter is omitted and monochrome attributes are to be used, no initial attribute is written to the display for the data.

The monochrome attribute and color attribute parameters consist of 2 bytes each: an initial and ending screen attribute to be used for a monochrome or a color display, respectively. One of these parameters will be selected based on the underlying display type, and the other will be discarded. Any of the attributes can be specified as a special value, X'00', indicating that no screen attribute should be written to the display. If the initial screen attribute is specified as an actual attribute, the data column, if specified, must be greater than or equal to 2. The initial screen attribute, if not X'00', will be written to the screen at the column previous to the first data character if row and column are specified, otherwise to the current display address. The ending screen attribute, if not X'00', will be written at the column directly after the last data character. See "Screen Attribute Characters" on page 14-3 for a description of the screen attribute values.

Ending monochrome attribute
INPUT; CHAR(1)
The ending screen attribute for monochrome displays. If this parameter is omitted, and monochrome attributes are to be used, no ending attribute is written to the display for the data.

Starting color attribute
INPUT; CHAR(1)
The initial screen attribute for color displays. If this parameter is omitted and color attributes are to be used, no initial attribute is written to the display for the data.

Ending color attribute
INPUT; CHAR(1)
The ending screen attribute for color displays. If this parameter is omitted and color attributes are to be used, no ending attribute is written to the display for the data.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the data is written to the screen at the specified location. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA308 E Attempt to write data past end of display.
CPFA30D E Invalid screen attribute.
CPFA31D E Attempt to write outside of window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA333 E Parameter &1 not positive integer value.
CPFA334 E Low-level environment handle incorrect.
CPFA335 E Screen address parameter error.
CPFA33C E Undefined field ID &1.
CPFA33F E CPFA33F

Write Structured Field Major (QsnWrtSFMaj) API

Parameters

Required Parameter Group:

1	Major structure	Input	Char(*)
2	Major structure length	Input	Binary(4)

Optional Parameter Group:

3	Field ID	Input	Binary(4)
4	Row	Input	Binary(4)
5	Column	Input	Binary(4)
6	Command buffer handle	Input	Binary(4)
7	Low-level environment handle	Input	Binary(4)
8	Error code	I/O	Char(*)

Returned Value:

Return code	Output	Binary(4)

The Write Structured Field Major (QsnWrtSFMaj) API writes the major structure of a structured field, setting the display address to the given row and column. If a command buffer is specified that does not contain a previous or current WTD operation, one is implicitly added to the buffer using the control characters QSN_CC1_NULL and QSN_CC2_UNLOCKBD.

Use this API in conjunction with the Write Structured Field Minor (QsnWrtSFMin) API to construct a structured field operation. For indirect operations, the length contained in the minor structure data parameter is added to the stored length for this major structure for every indirect QsnWrtSFMin operation encountered directly after this operation. In this way, you need only calculate the length of each individual structure for constructing a structured field operation. See the *5494 Remote Control Unit Functions Reference* for more information regarding structured fields.

This command corresponds indirectly to the 5250 Write to Display (WTD) command with a Set Buffer Address and a Write to Display Structured Field order if the row and column are specified. (For an indirect operation, a WTD is placed in the command buffer only if one does not already exist in the buffer.)

Restrictions

If window mode is not set and the data (including attributes) exceeds the length of a row, the data will be wrapped to the next line. If bottom-to-top wrapping is not supported, a CPFA308 error occurs when the data (including attributes) exceeds the last row on the screen.

Required Parameter Group

Not all structured field types are supported by all devices. A negative response code is issued if an attempt is made to write a type to a device that does not support it.

Major structure
INPUT; CHAR(*)
The major structure to be written to the screen. The data must consist of the entire major structure as documented in the *5494 Remote Control Unit Functions Reference*.

Major structure length
INPUT; BINARY(4)
The length of the major structure parameter. This is the length only and does not include any associated minor structure lengths.

Optional Parameter Group

Field ID
INPUT; BINARY(4)
The field ID indicating the field at which to set the display address. If this parameter is specified with a nonzero value, the row and column parameters are ignored and the row and column values corresponding to the field ID are used to set the display address. If neither the field ID nor the row and column parameters are specified, the current display address is used.

Row
INPUT; BINARY(4)
The row at which to write the structure. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the row location of the top window border (0 for full screen) if `offset` is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if `offset` is negative. The `offset` is the row parameter value specified, and `actual` is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

If both row and column are omitted, the data is written starting at the current display address. If this is the case and the command is a direct operation, or the buffer specified does not contain a preceding output operation that sets the display address, the current display address is set to row 1, column 1 prior to writing the data.

Row and column must both be specified or omitted; one cannot be specified if the other is omitted.

Column
INPUT; BINARY(4)
The column at which to write the data. The column parameter must refer to a column no greater than the current screen or window mode width (if window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the column location of the left window border (0 for full screen) if `offset` is positive, or the column location of the right window border (screen width plus 1 for full screen) if `offset` is negative. The `offset` is the column parameter value specified, and `actual` is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the data is written to the screen at the specified location. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA306 E Command not supported by current device.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA308 E Attempt to write data past end of display.
CPFA31D E Attempt to write outside of window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA333 E Parameter &1 not positive integer value.
CPFA334 E Low-level environment handle incorrect.
CPFA335 E Screen address parameter error.
CPFA341 E Length &2 of structure incorrect.
CPFA33C E Undefined field ID &1.

Write Structured Field Minor (QsnWrtSFMin) API

Parameters

Required Parameter Group:

1	Minor structure	Input	Char(*)
2	Minor structure length	Input	Binary(4)
3	Command buffer handle	Input	Binary(4)

Optional Parameter Group:

4	Low-level environment handle	Input	Binary(4)
5	Error code	I/O	Char(*)

Returned Value:

Return code	Output	Binary(4)

The Write Structured Field Minor (QsnWrtSFMin) API writes the minor structure of a structured field, incrementing the length field in the corresponding major structure by the length of this minor structure. The QsnWrtSFMin API must directly follow a QsnWrtSFMaj or QsnWrtSFMin operation to the given command buffer.

Use this API in conjunction with the Write Structured Field Major (QsnWrtSFMaj) API to construct a structured field operation. For indirect operations, the length contained in the minor structure data parameter is added to the stored length for this major structure for every indirect QsnWrtSFMin operation encountered directly after this operation. In this way, you need only calculate the length of each individual structure for constructing a structured field operation. See the *5494 Remote Control Unit Functions Reference* for more information regarding structured fields.

Restrictions

Not all structured field types are supported by all devices. A negative response code is issued if an attempt is made to write a type to a device that does not support it.

Required Parameter Group

Minor structure
INPUT; CHAR(*)
The minor structure to be written to the screen. The data must consist of the entire minor structure as documented in the *5494 Remote Control Unit Functions Reference*.

Minor structure length
INPUT; BINARY(4)
The length of the minor structure.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. The last operation stored in this command buffer must have been either a QsnWrtSFMaj or QsnWrtSFMin operation.

Optional Parameter Group

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA308 E Attempt to write data past end of display.
CPFA30D E Invalid screen attribute.
CPFA31D E Attempt to write outside of window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA333 E Parameter &1 not positive integer value.
CPFA334 E Low-level environment handle incorrect.
CPFA335 E Screen address parameter error.
CPFA33B E Incorrect operation before QsnWrtSFMin.
CPFA33C E Undefined field ID &1.
CPFA341 E Length &2 of structure incorrect.

Write to Display (QsnWTD) API

Parameters

Required Parameter Group:			
1	Control character byte 1	Input	Char(1)
2	Control character byte 2	Input	Char(1)
Optional Parameter Group:			
3	Command buffer handle	Input	Binary(4)
4	Low-level environment handle	Input	Binary(4)
5	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Write to Display (QsnWTD) API issues a Write to Display (WTD) command. The display address is not affected by this command. A WTD command is used to indicate the beginning of a series of output operations. For most of the DSM screen output operations, the control unit requires that a WTD be issued prior to the actual operation itself, in the same I/O operation. Multiple output operations can be sent with a given WTD command by using a command buffer. For direct operations, such output APIs, implicitly issue a WTD command. For indirect operations, a WTD is added to the buffer if one does not exist already. Each time a WTD command is received, the control unit implicitly sets the display address to row 1, column 1, prior to issuing the operation.

For example, if two direct QsnWrtDta operations are performed in succession and the second operation does not specify the row and column parameters, the data is written to row 1, column 1, and not the display address following the first QsnWrtDta operation. If two indirect QsnWrtDta operations are issued to an empty command buffer, the first QsnWrtDta operation causes a WTD command to be added to the buffer, but the second QsnWrtDta operation can use the existing command. In this case, if no row and column are specified, the display address used for the second operation will be the one after the first operation is issued. If an intervening operation that does not require a WTD, such as a QsnClrScr or a QsnSetErr, is added to the command buffer between two operations that do require a WTD command, a second WTD command is added to the buffer for the second operation.

The screen output APIs that require a WTD command and that correspond to the WTD command **orders** (as described in *5250 Functions Reference*) are as follows:

- Set Output Address (QsnSetOutAdr)
- Write Data (QsnWrtDta)
- Write Transparent Data (QsnWrtTDta)
- Pad for N Positions (QsnWrtPad)
- Pad between Two Screen Addresses (QsnWrtPadAdr)
- Set Field (QsnSetFld)
- Set Cursor Address (QsnSetCsrAdr)
- Insert Cursor (QsnInsCsr)

The cursor position is not affected if the keyboard is unlocked when command processing begins and is not locked during command processing, or if a parameter error is detected. If specified by control character byte 1, the cursor will be moved to the insert cursor or default location when the keyboard is unlocked.

This operation corresponds directly to the 5250 Write to Display command. If this is an indirect operation, this operation will issue a new WTD command to the command buffer.

Required Parameter Group

Control character byte 1
INPUT; CHAR(1)
The operation for the display to perform prior to processing the Write to Display command. See "Control Characters" on page 14-2 for a description of the control character values.

Control character byte 2
INPUT; CHAR(1)
The operation for the display to perform after the Write to Display command has been processed. See "Control Characters" on page 14-2 for a description of the control character values.

Optional Parameter Group

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is omitted or specified as 0, this is a direct operation and the command is sent to the display. Otherwise, this is an indirect operation and the command is stored in the command buffer without an I/O operation taking place.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The

value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA303 E Error occurred during screen I/O operation.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA31C E Incorrect value for control character byte &1.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA334 E Low-level environment handle incorrect.

Write Transparent Data (QsnWrtTDta) API

Parameters

Required Parameter Group:

1	Data	Input	Char(*)
2	Data length	Input	Binary(4)

Optional Parameter Group:

3	Field ID	Input	Binary(4)
4	Row	Input	Binary(4)
5	Column	Input	Binary(4)
6	Starting monochrome attribute	Input	Char(1)
7	Ending monochrome attribute	Input	Char(1)
8	Starting color attribute	Input	Char(1)
9	Ending color attribute	Input	Char(1)
10	Command buffer handle	Input	Binary(4)
11	Low-level environment handle	Input	Binary(4)
12	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Write Transparent Data (QsnWrtTDta) API writes *n* bytes of transparent data to the display.

This command allows transmission of data with any value (X'00' to X'FF') to the display screen. If the data destination is a 5250 display, and if the data X'04', X'11', or X'FF' is transmitted, unpredictable results occur. Note that if DBCS characters are included in the data, the host system does not perform IGC extension character processing. However, SI/SO characters will be processed correctly.

The display address after this operation is one position past the last data byte written to the screen.

This command corresponds indirectly to the 5250 Write to Display (WTD) command with a Set Buffer Address and a Transparent Data order. (For an indirect operation, a WTD is placed in the command buffer only if one does not already exist in that buffer.)

Restrictions

This command is not supported by all control units. A CPFA306 error occurs if an attempt is made to issue this command to a control unit that does not support it. Additional restrictions apply as for the Write Data (QsnWrtDta) API.

Required Parameter Group

Data
INPUT; CHAR(*)
The data to be written to the screen.

Data length
INPUT; BINARY(4)
The number of bytes of data to be written.

Optional Parameter Group

Field ID
INPUT; BINARY(4)
The field ID indicating the field at which to set the display address. If this parameter is specified with a nonzero value, the row and column parameters are ignored and the row and column values corresponding to the field ID are used to set the display address. If neither the field ID nor the row and column parameters are specified, the current display address is used.

Row
INPUT; BINARY(4)
The row at which to write the data. The row parameter must refer to a row no greater than the current screen or window mode height (if window mode is enabled). The actual screen row used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the row location of the top window border (0 for full screen) if `offset` is positive, or the row location of the bottom window border (screen height plus 1 for full screen) if `offset` is negative. The `offset` is the row parameter value specified, and `actual` is the actual screen row to be used. A CPFA307 error occurs if an incorrect row value is specified.

If both row and column are omitted, the data is written starting at the current display address. If this is the case and the command is a direct operation, or the buffer specified does not contain a preceding output operation that sets the display address, the current display address is set to row 1, column 1, prior to writing the data. Row and column must both be specified or omitted; one cannot be specified if the other is omitted.

Column
INPUT; BINARY(4)
The column at which to write the data. The column parameter must refer to a column no greater than the current screen or window mode width (if window mode is on). The actual screen column used for a screen I/O operation is calculated using the formula `base+offset=actual`. The `base` is the column location of the left window border (0 for full screen) if `offset` is positive, or the column location of the right window border (screen width plus 1 for full screen) if `offset` is negative. The `offset` is the column parameter value specified, and `actual` is the actual screen column to be used. A CPFA307 error occurs if an incorrect column value is specified.

Starting monochrome attribute
INPUT; CHAR(1)
The initial screen attribute for monochrome displays. If this parameter is omitted and monochrome attributes are to be used, no initial attribute is written to the display for the data. See "Screen Attribute Characters" on page 14-3 for a description of the screen attribute values. The attribute parameters are specified with the same effect as for the QsnWrtDta operation.

Ending monochrome attribute
INPUT; CHAR(1)
The ending screen attribute for monochrome displays. If this parameter is omitted and monochrome attributes are to be used, no ending attribute is written to the display for the data.

Starting color attribute
INPUT; CHAR(1)
The initial screen attribute for color displays. If this parameter is omitted and color attributes are to be used, no initial attribute is written to the display for the data.

Ending color attribute
INPUT; CHAR(1)
The ending screen attribute for color displays. If this parameter is omitted and color attributes are to be used, no ending attribute is written to the display for the data.

Command buffer handle
INPUT; BINARY(4)
A handle for the command buffer in which to store the command. If this parameter is specified, this is an indirect operation; the command is stored in the command buffer without an I/O operation taking place. If this parameter is omitted or specified with a zero value, this is a direct operation; the data is written to the screen at the specified location.

Low-level environment handle
INPUT; BINARY(4)
The low-level environment that the operation applies to. If this parameter is omitted or given with a value of zero, the default low-level environment is used.

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.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA301 E Command buffer is full.
CPFA304 E Data-stream error &1 reported for screen I/O operation.
CPFA305 E Cannot add operation to command buffer.
CPFA306 E Command not supported by current device.
CPFA307 E Screen position &1,&2 outside of display or window area.
CPFA31E E Required parameter &1 omitted.
CPFA31D E Attempt to write outside of window area.
CPFA31E E Required parameter &1 omitted.
CPFA331 E Buffer handle incorrect.
CPFA333 E Parameter &1 not positive integer value.
CPFA334 E Low-level environment handle incorrect.
CPFA335 E Screen address parameter error.
CPFA33C E Undefined field ID &1.

Low-Level Services Examples

The sample ILE C/400 program in Figure 18-3 on page 18-25 shows how to use the Write Data (QsnWrtDta), Get AID (QsnGetAID), and Roll Up (QsnRollUp) APIs.. The program writes a line at the bottom of the screen and if F3 is not pressed, rolls the screen up and writes a new line at the bottom of the screen. The roll area for the QsnRollUp API is defined to be from row 1 to 24 and one line is rolled up. If F3 is pressed, the program ends. A partial screen display is shown in Figure 18-4 on page 18-25.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "qsnll.h"

int main(void)
{
    long i;
    char s[100];

    QsnClrScr('0', 0, 0, NULL);
    for (i = 1; ; ++i) {
        sprintf(s, "Line %2.d.   Press Enter to roll, F3 to quit.", i);
        QsnWrtDta(s, strlen(s), 0, 24, 2, QSN_SA_NORM, QSN_SA_NORM,
                  QSN_SA_NORM, QSN_SA_NORM, 0, 0, NULL);
        if (QsnGetAID(NULL, 0, NULL) == QSN_F3)
            break;
        QsnRollUp(1, 1, 24, 0, 0, NULL);
    }
}
```

Figure 18-3. Program to Roll Text on Screen

```
Line 1.   Press Enter to roll, F3 to quit.
Line 2.   Press Enter to roll, F3 to quit.
Line 3.   Press Enter to roll, F3 to quit.
Line 4.   Press Enter to roll, F3 to quit.
Line 5.   Press Enter to roll, F3 to quit.
Line 6.   Press Enter to roll, F3 to quit.
Line 7.   Press Enter to roll, F3 to quit.
Line 8.   Press Enter to roll, F3 to quit.
```

Figure 18-4. Screen Output from Roll Text Program

Figure 18-5 on page 18-26 shows how to use the Query 5250 (QsnQry5250) API. A sample display is shown in Figure 18-6 on page 18-26.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "qsnll.h"

#define    TRUE    1
#define    FALSE   0

int main(void)
{
    Qsn_Qry_5250_T qry5250;
    Qsn_WSC_display_T *dsp = (Qsn_WSC_display_T *)(qry5250.WSC_display);
    char s[100];
    Qsn_Cmd_Buf_T buf;

    QsnQry5250(&qry5250, sizeof(qry5250), NULL);
    if ( (buf=QsnCrtCmdBuf(100, 20, 0, NULL, NULL)) == Q_NO_HANDLE) {
        printf("Q5250: command buffer create failed");
        exit(QSN_FAIL);
    }
    QsnClrScr('0', buf, Q_NO_HANDLE, NULL);
    sprintf(s, "Query status is %c, num input fields: %d, color: %c, wide: %c",
            qry5250.query_status, qry5250.num_input_capable,
            (QsnQryColorSup(NULL, Q_NO_HANDLE, NULL) == TRUE ? 'Y' : 'N'),
            (dsp->scr_size == 3 ? 'Y' : 'N'));
    QsnWrtDta(s, strlen(s), 0, 5, 5, QSN_SA_NORM, QSN_SA_NORM,
              QSN_SA_NORM, QSN_SA_NORM, buf, Q_NO_HANDLE, NULL);
    sprintf(s, "GUI display: %d, GUI support: %d",
              dsp->GUI_display, dsp->GUI_support);
    QsnWrtDta(s, strlen(s), 0, 7, 5, QSN_SA_NORM, QSN_SA_NORM,
              QSN_SA_NORM, QSN_SA_NORM, buf, Q_NO_HANDLE, NULL);
    sprintf(s, "Move cursor: %d, Extended primary attributes: %d",
              dsp->move_csr_order, dsp->extended_pri_atr_DP);
    QsnWrtDta(s, strlen(s), 0, 9, 5, QSN_SA_NORM, QSN_SA_NORM,
              QSN_SA_NORM, QSN_SA_NORM, buf, Q_NO_HANDLE, NULL);
    sprintf(s, "Wide mode on: %c",
              QsnRtvMod(NULL, Q_NO_HANDLE, NULL) == QSN_DSP04 ? 'y' : 'n');
    QsnWrtDta(s, strlen(s), 0, 11, 5, QSN_SA_NORM, QSN_SA_NORM,
              QSN_SA_NORM, QSN_SA_NORM, buf, Q_NO_HANDLE, NULL);
    sprintf(s, "Wide mode allowed: %c",
              QsnQryModSup(QSN_DSP04, NULL, Q_NO_HANDLE, NULL) ==
                                                  TRUE ? 'y' : 'n');
    QsnWrtDta(s, strlen(s), 0, 13, 5, QSN_SA_NORM, QSN_SA_NORM,
              QSN_SA_NORM, QSN_SA_NORM, buf, Q_NO_HANDLE, NULL);
    QsnPutBuf(buf, Q_NO_HANDLE, NULL);
    QsnGetAID(NULL, Q_NO_HANDLE, NULL);
}
```

Figure 18-5. Program Using the Query 5250 (QsnQry5250) API

```
Query status is 1, num input fields: 500, color: Y, wide: Y
GUI display: 1, GUI support: 1
Move cursor: 0, Extended primary attributes: 0
Wide mode on: n
Wide mode allowed: y
```

Figure 18-6. Screen Output from QsnQry5250 API Program

The sample program in Figure 18-7 on page 18-27 shows how to use the Read Modified Fields (QsnReadMDT) API in conjunction with the buffer query APIs. The resulting screen display before and after the input operations is shown in Figure 18-8 on page 18-28 and Figure 18-9 on page 18-28, respectively.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "qsnll.h"

#define    TRUE     1
#define    FALSE    0

int main(void)
{
    Q_Bin4 i, t1, t2, t3, numflds;
    const Q_Uchar cc1=QSN_CC1_MDTALL_CLRALL, cc2=QSN_CC2_UNLOCKBD;
    const Q_Uchar nosa = QSN_NO_SA, norm = QSN_SA_NORM, saul = QSN_SA_UL;
    char pad = ' ';
    Qsn_Cmd_Buf_T cmdbuf, cmdbuf2;
    Qsn_Inp_Buf_T inpbuf;
    Qsn_Fld_Inf_T fldqry;
    Qsn_Read_Inf_T rdqry;
    char msg[100];
    Q_Fdbk_T fdbk = { sizeof(Q_Fdbk_T) };

    cmdbuf = QsnCrtCmdBuf(100, 50, 0, NULL, NULL);
    cmdbuf2 = QsnCrtCmdBuf(100, 50, 0, NULL, NULL);
    inpbuf = QsnCrtInpBuf(200, 50, 0, NULL, NULL);
    QsnClrScr('0', 0, 0, NULL);
    QsnWTD(cc1, cc2, cmdbuf, 0, NULL);
    while (TRUE) {
        QsnSetFld(0, 10, 3, 2, QSN_FFW_ALPHA_SHIFT, NULL, 0, saul,
                    saul, cmdbuf, 0, NULL);
        QsnSetFld(0, 10, 5, 2, QSN_FFW_ALPHA_SHIFT, NULL, 0, saul,
                    saul, cmdbuf, 0, NULL);
        QsnSetFld(0, 10, 7, 2, QSN_FFW_ALPHA_SHIFT, NULL, 0, saul,
                    saul, cmdbuf, 0, NULL);
        numflds = QsnReadMDT(QSN_CC1_NULL, QSN_CC1_NULL, NULL,
                             inpbuf, cmdbuf, 0, NULL);
        if (QsnRtvReadAID(inpbuf, NULL, NULL) == QSN_F3)
            break;
        QsnPutBuf(cmdbuf2, 0, NULL);
        QsnClrBuf(cmdbuf2, NULL);
        QsnClrBuf(cmdbuf, NULL);
        QsnWTD(cc1, cc2, cmdbuf, 0, NULL);
        sprintf(msg, "Num Fields Change: %d", numflds);
        QsnWrtDta(msg, strlen(msg), 0, 2, 30, norm, norm, norm, norm,
                  cmdbuf, 0, NULL);
        for (i = 1; i <= numflds; i++) {
            fldqry.len = 0;
            if (QsnRtvFldInf(inpbuf, i, &fldqry, sizeof(fldqry),
                                0, &fdbk) != QSN_FAIL) {
                sprintf(msg, "Field# %d, row %d, col %d, len %d, value %.*s",
                       i, fldqry.row, fldqry.col, fldqry.len,
                       fldqry.len, fldqry.data);
                QsnWrtDta(msg, t1=strlen(msg), 0,
                     t2=i+3, t3=30, norm, norm, norm, norm,
                     cmdbuf, 0, NULL);
                QsnWrtPad(pad, t1, 0, t2, t3, cmdbuf2, 0, NULL);
            } else {
                sprintf(msg, "Field query failed");
                QsnWrtDta(msg, t1=strlen(msg), 0,
                     t2=4, t3=30, norm, norm, norm, norm,
                     cmdbuf, 0, NULL);
                QsnWrtPad(pad, t1, 0, t2, t3, cmdbuf2, 0, NULL);
            }
        }
```

Figure 18-7 (Part 1 of 2). Program Using Read Modified Fields (QsnReadMDT) API

```
        QsnRtvReadInf(inpbuf, &rdqry, sizeof(rdqry), 0, NULL);
        sprintf(msg, "Read information:");
        QsnWrtDta(msg, strlen(msg), 0, t2=10, t3=2, norm, norm, norm, norm,
                  cmdbuf, 0, NULL);
        QsnWrtPadAdr(pad, -1, -1, t2, t3, cmdbuf2, 0, NULL);
        sprintf(msg, "Bytes returned %d, available: %d",
                rdqry.bytes_returned, rdqry.bytes_available);
        QsnWrtDta(msg, strlen(msg), 0, ++t2, t3+2, norm, norm, norm, norm,
                  cmdbuf, 0, NULL);
        sprintf(msg, "First data byte: %p", rdqry.dta);
        QsnWrtDta(msg, strlen(msg), 0, ++t2, t3+2, norm, norm, norm, norm,
                  cmdbuf, 0, NULL);
        sprintf(msg, "First field byte: %p", rdqry.fld_dta);
        QsnWrtDta(msg, strlen(msg), 0, ++t2, t3+4, norm, norm, norm, norm,
                  cmdbuf, 0, NULL);
        sprintf(msg, "Diff: %d", rdqry.fld_dta-rdqry.dta);
        QsnWrtDta(msg, strlen(msg), 0, ++t2, t3+4, norm, norm, norm, norm,
                  cmdbuf, 0, NULL);
        sprintf(msg, "Read len: %d, data len: %d, field data len: %d, num fields: %d",
                rdqry.read_len, rdqry.dta_len, rdqry.fld_dta_len, rdqry.num_flds);
        QsnWrtDta(msg, strlen(msg), 0, ++t2, t3+2, norm, norm, norm, norm,
                  cmdbuf, 0, NULL);
        sprintf(msg, "Read row: %d, col: %d, aid: %x",
                rdqry.read_row, rdqry.read_col, rdqry.AID);
        QsnWrtDta(msg, strlen(msg), 0, ++t2, t3+2, norm, norm, norm, norm,
                  cmdbuf, 0, NULL);
    }
}
```

Figure 18-7 (Part 2 of 2). Program Using Read Modified Fields (QsnReadMDT) API

```
field 1
  field 2
field 3
```

Figure 18-8. Display Screen before Input Operation

```
                          Num Fields Change: 3
_______                    Field# 1, row 3, col 2, len 7, value field 1
_______                    Field# 2, row 5, col 2, len 9, value field 2
_______                    Field# 3, row 7, col 2, len 7, value field 3


Read information:
  Bytes returned 80, available: 80
  First data byte: SPP:0000 :1aefQPADEV0010JENNIFER  002600 :19b:0:d42
    First field byte: SPP:0000 :1aefQPADEV0010JENNIFER  002600 :19e:1:d42
    Diff: 3
  Read len: 35, data len: 35, field data len: 32, num fields: 3
  Read row: 3, col: 2, aid: f1
```

Figure 18-9. Display Screen after Input Operation

Performance Considerations

The following operations can incur overhead and adversely affect the performance of your application:

- **QsnCrtEnv**

  Specifying translation of x'3f' to x'1f' can incur overhead, because all outgoing data must be checked for this value. This option should be specified only if CDRA is on and translation between the code pages will result in a x'3f' occurrence in data to be displayed.

- **QsnSavScr**

  This operation results in the entire contents of the screen being read, which can adversely affect response time. This is typically about 3KB, but could be up to 28KB.

- **QsnRstScr**

  This operation writes the result of a save screen back to the device, which can adversely affect response time.

- If you have GUI support, you can put additional commands after the QsnSavScr or QsnRstScr APIs to reduce I/O operations and improve performance.

- Deleting structures associated with handles, such as command buffers, prior to exiting a program will improve performance for the programs that use APIs that create handles.

Limitations and Restrictions

The following limitations or restrictions apply to the low-level interfaces:

- Certain functions are supported by control units that do not support the 5250 Query command. If the Query command is not supported, it is assumed that the particular function is not supported either. These functions are transparent data support and move cursor order support. Device attributes such as wide mode and color support will be determined based on the device type if the 5250 Query command is not supported.
- In order for the Retrieve Display Mode (QsnRtvMod) operation to correctly report the current state of the display, all commands that affect this state (like a Clear Unit or Clear Unit Alternate) must occur as the first command in any command stream written to the display. This is because the work station control unit inspects the first command in the stream to determine if a state change is taking place. Most AS/400 programs, including the DSM APIs, only send these commands at the beginning of a stream. If you write a stream in which such commands do not appear at the beginning of the stream, the result of the Retrieve Display Mode (QsnRtvMod) operation may not be accurate.
- When conversions are performed, they are performed only after a Read Input Fields (QsnReadInp), Read Modified Fields (QsnReadMDT), Read Modified Alternate (QsnReadMDTAlt), Read Immediate (QsnReadImm), or Read Modified Immediate Alternate (QsnReadMDTImmAlt) operation. They are performed on all incoming field data, including transparent and numeric data. You must turn conversion on and off. To prevent certain data from being converted, you explicitly set the conversion options on the QsnCrtEnv and QsnChgEnv APIs. The conversions that are affected by this are CDRA conversion based on the job CCSID and conversions of X'1F' in the incoming data stream to X'3F'.

Chapter 19. Introduction to the Window Services APIs

The window services APIs consist of two functional groups: the window builder and window manager services. The window builder APIs provide the services needed to create, delete, move, and resize windows. The window builder services include the window manipulation and query APIs, and the window I/O APIs. The window manager APIs provide the services needed to manage multiple windows, support I/O to several active windows, and switch between windows.

A window is created using a window and a low-level environment description. The window description provides the window attributes, a pointer to data that is specified by the using application, and several exit routines that the window module calls when a window is moved, resized, or deleted so the using program can perform the appropriate actions. The exit programs are called after the window has been modified and redrawn, or prior to deallocating the window. The low-level environment description is the same as that used on the Create Low-Level Environment (QsnCrtEnv) API to create a low-level environment. A window is implemented as a low-level environment where the user data pointer describes the window itself. Thus, a window can be manipulated through the low-level APIs or through the window APIs by using the same window handle. This implementation is similar to the concept of inheritance in object-oriented programming languages. DSM window support uses graphical user interface (GUI) when the underlying control unit supports it.

Each window has the low-level environment window mode enabled. The low-level environment window area is set to the usable area in the window, which consists of the area inside the border and attributes that can be accessed by screen I/O services. (It does not include the message line.) Use relative coordinates when specifying a row and column on an I/O API. The upper left corner of the usable area is (1,1). To use absolute coordinates with a window, disable the low-level environment window mode with the Set Low-Level Environment Window Mode (QsnSetEnvWinMod) API.

Figure 19-1 on page 19-2 shows the components of a DSM window. The window in this example has a specified depth of 13 rows and a width of 19 columns.

The attributes of a DSM window are similar to those of a data description specifications (DDS) window. The initial size and location of a DSM window are specified using the location of the upper-left window border character and the number of rows and columns within the window. For DSM windows, the leading window attribute, right continuation attribute, or message line can be specified separately. Unlike a DDS window, a DSM window does not require the following:

- A border

  If a window is defined with no border, no extra space is used on the display for the border characters or their attributes (L and B in Figure 19-1 on page 19-2). An area of the screen is cleared starting from the specified location for the upper left corner of the window and continues for the number of rows and columns given as the window size. Figure 19-2 on page 19-2 shows an example of a window with no border.

  Note: In discussions throughout the DSM sections that refer to window borders, this should be taken to mean the top/bottom window row or left/right window column for a window with no borders.

- Window border attributes

  If a window is defined with no border attributes (L and B in Figure 19-1 on page 19-2), no extra space for these is used on the display. Figure 19-3 on page 19-2 shows a window with no border attributes.

- Leading window attribute

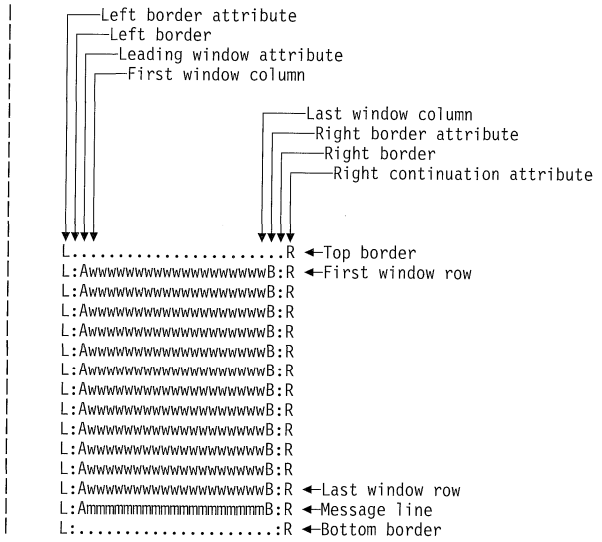
  The leading window attribute (A in Figure 19-1 on page 19-2) is part of the addressable window area. If specified, this attribute takes up an extra screen character and does not reduce the size of the window area. Figure 19-4 on page 19-2 shows a window defined with no leading window attribute. The window text directly follows the window border character. Attribute characters can be written inside the window if desired.

- Right continuation attribute

  If the right continuation attribute (R in Figure 19-1 on page 19-2) is specified on a window description, DSM determines the appropriate attribute to use, based on the screen image underlying the window. If the window is not a GUI window, right-continuation-attribute correction is performed for display-screen and presentation screen DBCS-only attributes. (No correction is performed for extended primary attributes and DBCS data types). When a window is placed on a screen that supports extended attributes, all extended attributes are cleared prior to displaying the window. To have the right continuation attribute handle extended attributes, you must use a display that supports GUI windows and specify GUI window support on the window description.

- Message line

  Specifying a message line on a window description decreases the number of lines in the window area by 1, as shown in Figure 19-1 on page 19-2.



Key:

L	Left border attribute
.	Top and bottom border
R	Right continuation attribute
:	Left and right border
A	Leading window attribute
w	Window area
m	Message line
B	Right border attribute

Figure 19-1. DSM Window Layout

```
AwwwwwwwwwwwwwwwwwwR ←First window row
AwwwwwwwwwwwwwwwwwwR
AwwwwwwwwwwwwwwwwwwR
AwwwwwwwwwwwwwwwwwwR
AwwwwwwwwwwwwwwwwwwR
AwwwwwwwwwwwwwwwwwwR
AwwwwwwwwwwwwwwwwwwR
AwwwwwwwwwwwwwwwwwwR
AwwwwwwwwwwwwwwwwwwR
AwwwwwwwwwwwwwwwwwwR
AwwwwwwwwwwwwwwwwwwR ←Last window row
AmmmmmmmmmmmmmmmmmmR ←Message line
```

Figure 19-2. DSM Window with No Border

```
.....R ←Top border
:Awwwwwwwwwwwwwwwwwww:R ←First window row
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R
:Awwwwwwwwwwwwwwwwwww:R ←Last window row
:Ammmmmmmmmmmmmmmmmmm:R ←Message line
:.....:R ←Bottom border
```

Figure 19-3. DSM Window with No Border Attributes

```
L.....R ←Top border
L:wwwwwwwwwwwwwwwwwwwB:R ←First window row
L:wwwwwwwwwwwwwwwwwwwB:R
L:wwwwwwwwwwwwwwwwwwwB:R
L:wwwwwwwwwwwwwwwwwwwB:R
L:wwwwwwwwwwwwwwwwwwwB:R
L:wwwwwwwwwwwwwwwwwwwB:R
L:wwwwwwwwwwwwwwwwwwwB:R
L:wwwwwwwwwwwwwwwwwwwB:R
L:wwwwwwwwwwwwwwwwwwwB:R
L:wwwwwwwwwwwwwwwwwwwB:R
L:wwwwwwwwwwwwwwwwwwwB:R ←Last window row
L:mmmmmmmmmmmmmmmmmmmB:R ←Message line
L:.....:R ←Bottom border
```

Figure 19-4. DSM Window with No Leading Window Attribute

Chapter 20. Window Manipulation and Query APIs

The window manipulation and query APIs are:

Change Window (QsnChgWin) changes the description for a window.
Create a Window (QsnCrtWin) creates a window.
Initialize Window Description (QsnInzWinD) initializes a window description with default values.
Move Window (QsnMovWin) moves a window to a new screen location.
Move Window by User (QsnMovWinUsr) moves a window to a new screen location specified by the user.
Resize Window (QsnRszWin) changes the size of a window.
Resize Window by User (QsnRszWinUsr) changes the size of a window according to user-specified cursor movements.
Retrieve Window Data (QsnRtvWinDta) returns a pointer to the user data for the given window.
Retrieve Window Description (QsnRtvWinD) retrieves a copy of the description for a window.
Set Window Services Attributes (QsnSetWinAtr) sets the default attributes for the window services.

The detailed API descriptions are presented in alphabetical order.

Change Window (QsnChgWin) API

Parameters

Required Parameter Group:

1	Window handle	Input	Binary(4)
2	Window description	Input	Char(*)
3	Length of window description	Input	Binary(4)

Optional Parameter:

4	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Change Window (QsnChgWin) API changes the window description for the given window. The size cannot be changed for a window that contains DBCS data.

Required Parameter Group

Window handle
INPUT; BINARY(4)
A handle for the window that will have its description changed.

Window description
INPUT; CHAR(*)
The new window description for the given window. The format of the window description is shown in "Format of the Window Description" on page 20-3.

Length of window description
Input; BINARY(4)
The length of the window description parameter.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA314 E Memory allocation error.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3A1 E Window description value is incorrect.
CPFA3AA E Window handle incorrect.

Create a Window (QsnCrtWin) API

Parameters

Required Parameter Group:

1	Window description	Input	Char(*)
2	Length of window description	Input	Binary(4)

Optional Parameter Group:

3	User extension information	Input	Char(*)
4	Length of user extension information	Input	Binary(4)
5	Start window	Input	Char(1)
6	Low-level environment description	Input	Char(*)
7	Length of low-level environment description	Input	Binary(4)
8	Window handle	Output	Binary(4)
9	Error code	I/O	Char(*)

Returned Value:

	Window handle	Output	Binary(4)

The Create a Window (QsnCrtWin) API creates a window and returns a handle for that window. The window must be deleted using the Delete Low-Level Environment (QsnDltEnv) API. Whenever a window is made current, the window mode is set to the usable area of the window.

Required Parameter Group

Window description
INPUT; CHAR(*)
The window description defines the attributes for the window. The format of the window description is shown in "Format of the Window Description" on page 20-3.

Length of window description
INPUT; BINARY(4)
The length of the window description parameter.

Optional Parameter Group

User extension information
INPUT; CHAR(*)
The user extension information is used to associate data and exit routines with the window. This parameter is required if the length of user extension information parameter is supplied. This essentially enables the object-oriented programming concept of inheritance, allowing the window to be extended in a natural way. The user extension information cannot be changed once the window has been created. The format of this parameter is shown in "Format of the Window User Extension Information" on page 20-5.

Length of user extension information
INPUT; BINARY(4)
The length of the user extension information parameter.

Start window
Input; CHAR(1)
Whether the window should be displayed on the screen when it is allocated. The possible values are:

0 The window is not displayed on the screen when it is allocated. You must use the Start a Window (QsnStrWin) API to start the window.
1 The window is displayed on the screen when it is allocated. This is the default if this parameter is omitted.

Low-level environment description
INPUT; CHAR(*)
The low-level environment description defines the operating environment for low-level operations used to create and manipulate the window. This parameter is required if the length of the low-level environment description parameter is supplied. The format of the low-level environment description is shown in "Format of the Low-Level Environment Description" on page 15-4. If this parameter is omitted, a low-level environment will be created with default values.

Length of low-level environment description
INPUT; BINARY(4)
The length of the low-level environment description parameter.

Window handle
OUTPUT; BINARY(4)
The variable containing the handle for the window created after the QsnCrtWin API has completed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Window handle
OUTPUT; BINARY(4)
This API returns the value for the window handle parameter.

Restrictions

Windows where the associated low-level environment indicates DBCS support cannot be resized. GUI windows must fit within the screen boundary. This includes the leading left border attribute and the right border continuation attribute. GUI windows must have a border and the associated left and right border attributes for the left and right borders. The concept of current and noncurrent window border attributes does not apply to GUI windows. No error-checking is performed for GUI-specific fields. The fields are passed to the control unit, as specified, and any errors will be detected by the control unit.

Format of the Window Description

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Row location of upper left corner of window border
4	4	BINARY(4)	Column location of upper left corner of window border
8	8	BINARY(4)	Number of rows within the window
12	C	BINARY(4)	Number of columns within the window
16	10	BINARY(4)	Minimum number of rows within the window
20	14	BINARY(4)	Minimum number of columns within the window
24	18	BINARY(4)	Maximum number of rows within the window
28	1C	BINARY(4)	Maximum number of columns within the window
32	20	CHAR(1)	Full-screen flag for the window
33	21	CHAR(3)	Window display attributes for a monochrome display
36	24	CHAR(3)	Window display attributes for a color display
39	27	CHAR(1)	Border flag for the window
40	28	CHAR(1)	Border attributes flag for the window
41	29	CHAR(1)	Leading attribute flag for the window
42	2A	CHAR(1)	Right continuation attribute flag for the window
43	2B	CHAR(1)	Message line flag for the window
44	2C	CHAR(1)	Upper left border character
45	2D	CHAR(1)	Top border character
46	2E	CHAR(1)	Upper right border character
47	2F	CHAR(1)	Left border character
48	30	CHAR(1)	Right border character
49	31	CHAR(1)	Lower left border character
50	32	CHAR(1)	Bottom border character
51	33	CHAR(1)	Lower right border character
52	34	CHAR(1)	GUI support flag for the window
53	35	CHAR(1)	Flag byte 1 (GUI only)
54	36	CHAR(1)	Flag byte 2 (GUI only)
55	37	CHAR(1)	Reserved (GUI only). The default is X'00'.
56	38	CHAR(1)	Border flags (GUI only)
57	39	CHAR(1)	Window title flags (GUI only)
58	3A	CHAR(1)	Monochrome title attribute (GUI only)
59	3B	CHAR(1)	Color title attribute (GUI only)
60	3C	CHAR(1)	Reserved (GUI only). The default is X'00'.
61	3D	CHAR(3)	Reserved. This field must be set to X'00'.
64	40	BINARY(4)	Offset to title text (GUI only)
68	44	BINARY(4)	Length of title text (GUI only)
72	48	BINARY(4)	Reserved. This field must be set to X'00'.
76	4C	CHAR(*)	Title text (GUI only)

Field Descriptions

In the following descriptions, the default value refers to the value set by the Initialize Window Description (QsnInzWinD) API.

The GUI-only fields in the following descriptions refer to fields within the data stream for the Create Window command major and minor structures. See the *5494 Remote Control Unit Functions Reference* manual for details.

Border attributes flag for the window. Whether or not the window border has left and right border attributes. (See Figure 19-1 on page 19-2.) The possible values are:

0 Window border has no attributes.
1 Window border has attributes. This is the default.

Border flag for the window. This flag indicates whether or not the window has a border. (See Figure 19-1 on page 19-2.) The possible values are:

0 Window has no border.
1 Window has a border. This is the default.

If this field is set to 0, the border attributes are not written to the screen, regardless of the values of the other fields.

Border flags (GUI only). Determines if border presentation characters are used (byte 3 of the border presentation minor structure of the Create Window command). The default is X'80'.

Bottom border character. The character used for the bottom border. The default is X'00' for all border characters. The default border character used for non-GUI windows is '.'. For GUI windows, the default GUI character is used.

Color title attribute (GUI only). The display attribute to precede the title on a color display (byte 5 of the window title minor structure of the Create Window command). The default is X'20'.

Column location of upper left corner of window border. The column number of the upper left corner of the window. This must be a positive integer value greater than or equal to 0. For GUI windows, the minimum value must be 2. Otherwise, it must be a positive integer value greater than or equal to 0. For non-GUI windows, if 0 or 1 is specified, the left border of the window or the leading border attribute, respectively, will not be displayed on the screen unless the window is moved. The default value is such that the maximum-sized window will be displayed, with border and attributes, given the other window description attributes (for example, window border attributes). If the window is a full-screen window, this field is ignored.

Flag byte 1 (GUI only). Byte 5 of the Create Window command major structure. The default is X'00'.

Flag byte 2 (GUI only). Byte 6 of the Create Window command major structure. The default is X'00'.

Full-screen flag for the window. Indicates whether or not this is a full-screen window. (A full-screen window cannot be moved or resized.) The possible values are:

0 Window is not a full-screen window. This is the default.
1 Window is a full-screen window.

GUI support flag for the window. This flag indicates whether or not GUI support should be used to build the window if the underlying device supports it. GUI windows always include a leading and trailing border attribute. The possible values are:

0 Do not use GUI support.
1 Use GUI support if the underlying device supports it. This is the default.

Leading attribute flag for the window. This flag indicates whether or not the window has a leading attribute. (See Figure 19-1 on page 19-2.) The possible values are:

0 Window has no leading attribute byte.
1 Window has a leading attribute byte. This is the default.

Left border character. The character used for the left border. The default is X'00' for all border characters. The default border character used for non-GUI windows is '.'. For GUI windows, the default GUI character is used.

Length of title text (GUI only). The length of the title text.

Lower left border character. The character used for the lower left border. The default is X'00' for all border characters. The default border character used for non-GUI windows is '.'. For GUI windows, the default GUI character is used.

Lower right border character. The character used for the lower right border. The default is X'00' for all border characters. The default border character used for non-GUI windows is '.'. For GUI windows, the default GUI character is used.

Maximum number of columns within the window. A value of 0, which is the default, indicates this value is the same as the maximum number of columns for the device in its current mode. If the window is a full-screen window, this field is ignored.

Maximum number of rows within the window. A value of 0, which is the default, indicates this value is the same as the maximum number of rows for the device in its current mode. If the window is a full-screen window, this field is ignored.

Message line flag for the window. This flag indicates whether or not the window has a message line. (See Figure 19-1 on page 19-2). The possible values are:

0 Window does not have a message line
1 Window has a message line. This is the default.

Minimum number of columns within the window. The minimum value allowed is 1. This is the default. If the window is a full-screen window, this field is ignored.

Minimum number of rows within the window. The minimum value allowed is 1. This is the default. If the window is a full-screen window, this field is ignored.

Monochrome title attribute (GUI only). The display attribute to precede the title on a monochrome display (byte 4 of the window title minor structure of the Create Window command). The default is X'20'.

Number of columns within the window. The number of columns in the window, from the first window column to the last. This excludes the left and right border, and the border and window attributes. (See Figure 19-1 on page 19-2.) A value of 0, which is the default, indicates this value is the maximum number of columns that can be defined given the other window description attributes (for example, window border attributes). The minimum allowed number of columns is 1. If the window is a full-screen window, this field is ignored.

Number of rows within the window. The number of rows in the window, from the first window row to the last. This includes the message line, if specified. (See Figure 19-1 on page 19-2.) A value of 0, which is the default, indicates this value is the maximum number of rows that can be defined given the other window description attributes (for example, window message line). The minimum allowed number of rows is 1. If the window is a full-screen window, this field is ignored.

Offset to title text (GUI only). The offset for the window title text.

Right border character. The character used for the right border. The default is X'00' for all border characters. The default border character used for non-GUI windows is '.'. For GUI windows, the default GUI character is used.

Right continuation attribute flag for the window. Whether or not the window has a right continuation attribute (see Figure 19-1 on page 19-2). The possible values are:

0 Window has no right continuation attribute byte.
1 Window has a right continuation attribute byte. This is the default.

The right continuation attribute used is X'20', which is green for color displays and normal attribute for monochrome displays.

Row location of upper left corner of window border. Specifies the row number of the upper left corner of the window. This must be a positive integer value greater than or equal to 0. For GUI windows, the minimum value must be 1. Otherwise, it must be a positive integer value greater than or equal to 0. For non-GUI windows, if 0 is specified, the top border of the window will not be displayed on the screen unless the window is moved. The default value is such that the maximum-sized window will be displayed, with border and attributes, given the other window description attributes (for example, window border attributes). If the window is a full-screen window, this field is ignored.

Title text (GUI only). The text for the window title (bytes 7 to *n* of the window title minor structure of the Create Window command).

Top border character. The character used for the top border. The default is X'00' for all border characters. The default border character used for non-GUI windows is '.'. For GUI windows, the default GUI character is used.

Upper left border character. The character used for the upper left border. The default is X'00' for all border characters. The default border character used for non-GUI windows is '.'. For GUI windows, the default GUI character is used.

Upper right border character. The character used for the upper right border. The default is X'00' for all border characters. The default border character used for non-GUI windows is '.'. For GUI windows, the default GUI character is used.

Window display attributes for a color display. The window display attributes for a color display. The first character is the attribute for the window border when the window is not current, the second for when the window is current, and the third for the leading window attribute. All bytes may contain the same value. The special value X'00' can be used to indicate that no screen attribute should be used for the given character. If X'00' is specified as the second attribute for a GUI window, the default GUI border attribute will be used. For each attribute field, both the monochrome and the color part must be either X'00' or a valid attribute. For example, it is incorrect to specify the second color attribute field as X'00' and the second monochrome attribute with a valid attribute.

The default values for these fields are those specified by the window services mode description (see "Format of the Window Services Attribute Description" on page 20-12).

Window display attributes for a monochrome display. The window display attributes for a monochrome display. The first character is the attribute for the window border when the window is not current, the second for when the window is current, and the third is for the leading window attribute. All bytes may contain the same value. The special value X'00' can be used to indicate that no screen attribute should be used for the given character. If X'00' is specified as the second attribute for a GUI window, the default GUI border attribute will be used. For each attribute field, both the monochrome and the color part must be either X'00' or a valid attribute. For example, it is incorrect to specify the second color attribute field as X'00' and the second monochrome attribute with a valid attribute.

The default values for these fields are those specified by the window services mode description (see "Format of the Window Services Attribute Description" on page 20-12).

Window title flags (GUI only). Byte 3 of the window title minor structure of the Create Window command. The default is X'00'.

Format of the Window User Extension Information

Offset		Type	Field
Dec	Hex		
0	0	PTR(SPP)	User data associated with window
16	10	PTR(PP)	Exit routine to call for Change Window (QsnChgWin) API
32	20	PTR(PP)	Exit routine to call for Delete Low-Level Environment (QsnDltEnv) API
48	30	PTR(PP)	Exit routine for Move Window (QsnMovWin), Move Window by User (QsnMovWinUsr), Resize Window (QsnRszWin), or Resize Window by User (QsnRszWinUsr) APIs
64	40	PTR(PP)	Exit routine for Display Window (QsnDspWin) API
80	50	PTR(PP)	Exit routine for Set Current Window (QsnSetCurWin) API

Field Descriptions

Exit routine for Change Window (QsnChgWin) API. This exit routine is called after the window is changed. If the window is redrawn, it is called after the window is redrawn.

Exit routine for Delete Low-Level Environment (QsnDltEnv) API. The exit routine to call when a window is deleted using the Delete Low-Level Environment (QsnDltEnv) API.

Exit routine for Display Window (QsnDspWin) API. The exit routine to call immediately before the window is drawn or redrawn. The following APIs may cause the window to be redrawn: QsnCrtWin, QsnStrWin, QsnChgWin, QsnMovWin, QsnMovWinUsr, QsnRszWin, QsnRszWinUsr, QsnDspWin, and QsnSetCurWin.

Exit routine for move or resize window APIs. The exit routine to call when window coordinates are changed using the Move Window (QsnMovWin), Move Window by User (QsnMovWinUsr), Resize Window (QsnRszWin), or Resize Window by User (QsnRszWinUsr) APIs. This exit routine is called after the window is redrawn.

Exit routine for Set Current Window (QsnSetCurWin) API. The exit routine to call whenever a window is made current by one of the following APIs: QsnCrtWin, QsnStrWin, or QsnSetCurWin. This exit routine is called after the window is drawn or redrawn.

User data associated with window. A pointer to any data that the user wants to associate with this window.

Window Exit Routines

Exit routines are user-supplied functions with a defined interface. The routines are called from certain APIs and allow the programmer to attach additional function to those APIs. For instance, if fields have been set up in a window, a Change Coordinates exit routine could be supplied to move the fields if the window is moved.

Change Window Exit Routine: This exit routine, if specified on the user extension information, is called when the window is changed. The following parameter is passed to the exit routine:

Parameter Passed to Exit Routine

1	Window handle	Input	Binary(4)

Change Window Exit Routine Parameter

Window handle
INPUT; BINARY(4)
The window that was changed.

Delete Window Exit Routine: This exit routine, if specified on the user extension information, is called when the window is deleted. The following parameter is passed to the exit routine:

Parameter Passed to Exit Routine

1	Window handle	Input	Binary(4)

Delete Window Exit Routine Parameter

Window handle
INPUT; BINARY(4)
The window that was deleted.

Change Window Coordinates Exit Routine: This exit routine, if specified on the user extension information, is called when the move or resize APIs are called, after the window has been successfully moved or resized. The following parameters are passed to the exit routine:

Parameters Passed to Exit Routine

1	Window handle	Input	Binary(4)
2	Top border offset	Input	Binary(4)
3	Left border offset	Input	Binary(4)
4	Bottom border offset	Input	Binary(4)
5	Right border offset	Input	Binary(4)

Change Window Coordinates Exit Routine Parameters

Window handle
INPUT; BINARY(4)
The window for which the coordinates were changed.

Top border offset
INPUT; BINARY(4)
The offset, in screen rows, from the previous top window border to the current top window border (after the window coordinates have been changed). It can be positive, negative, or zero, depending on how the top window border was changed. For example, if the top border was moved down two rows, this value would be 2; if it was moved up 4 rows, this value would be -4; if the top row was not changed, this value would be 0.

Left border offset
INPUT; BINARY(4)
The offset, in screen columns, from the previous left window border to the current left window border (after the window coordinates have been changed). It can be positive, negative, or zero, depending on how the left window border was changed. For example, if the left border was moved two columns to the right, this value would be 2; if it was moved 4 columns to the left, this value would be -4, and if the left column was not changed, this value would be 0.

Bottom border offset
INPUT; BINARY(4)
The offset, in screen rows, from the previous bottom

window border to the current bottom window border (after the window coordinates have been changed). It can be positive, negative, or zero, depending on how the bottom window border was changed. For example, if the border was moved down two rows, this value would be 2; if it was moved up 4 rows, this value would be -4; if the bottom row was not changed, this value would be 0.

Right border offset
INPUT; BINARY(4)
The offset, in screen columns, from the previous right window border to the current right window border (after the window coordinates have been changed). It can be positive, negative, or zero, depending on how the right window border was changed. For example, if the right border was moved two columns to the right, this value would be 2; if it was moved 4 columns to the left, this value would be -4; if the right column was not changed, this value would be 0.

Draw Window Exit Routine: This exit routine, if specified on the user extension information, is called when the Display Window (QsnDspWin) API is called, before the window is drawn. The following parameters are passed to the exit routine:

Parameters Passed to Exit Routine

1	Window handle	Input	Binary(4)
2	Command buffer	Input	Binary(4)

Draw Window Exit Routine Parameters

Window handle
INPUT; BINARY(4)
The window to be drawn.

Command buffer
INPUT; BINARY(4)
The command buffer used to store the commands that re-create the window contents. The contents of the command buffer are written to the screen along with the window border. This allows the window and its contents to be redrawn in a single I/O operation.

Current Window Exit Routine: This exit routine, if specified on the user extension information, is called when the window is made current through the Set Current Window (QsnSetCurWin) API. The following parameter is passed to the exit routine:

Parameter Passed to Exit Routine

1	Window handle	Input	Binary(4)

Current Window Exit Routine Parameter

Window handle
INPUT; BINARY(4)
A handle for the window that is made current.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA327 E Low-level environment description value incorrect.
CPFA314 E Memory allocation error.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA3A1 E Window description value is incorrect.
CPFA3AB E The value for parameter &1 must be 0 or 1.

Additional errors may be generated by this API. They are listed in "Error Messages" on page 15-6 under the Create Low-Level Environment (QsnCrtEnv) API.

Initialize Window Description (QsnInzWinD) API

Parameters

Required Parameter Group:

1	Window description	Output	Char(*)
2	Length of window description	Input	Binary(4)

Optional Parameter:

3	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Initialize Window Description (QsnInzWinD) API initializes a window description with default values. Unless otherwise specified in the window description (see "Format of the Window Description" on page 20-3), pointer fields are set to the null pointer, numeric fields to 0, character flag fields to 0, and other character fields to blanks. For example, the default value for the border flag is 1, so this field will be set to 1.

Required Parameter Group

Window description
OUTPUT; CHAR(*)
The window description to be initialized.

Length of window description
INPUT; BINARY(4)
The length of the window description parameter.

Optional Parameter Group

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.

Move Window (QsnMovWin) API

Parameters

Required Parameter Group:			
1	Window handle	Input	Binary(4)
2	Upper left row	Input	Binary(4)
3	Upper left column	Input	Binary(4)
Optional Parameter:			
4	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Move Window (QsnMovWin) API moves the window to the new upper left coordinate (upper left row, upper left column) specified. If the window can fit within the display at the location specified, it is moved to the new position. If a Change Window Coordinates exit routine is specified on the window description, it is called after the window is successfully moved.

Required Parameter Group

Window handle
INPUT; BINARY(4)
A handle for the window to be moved.

Upper left row
INPUT; BINARY(4)
The absolute screen row for the new upper left corner of the window.

Upper left col
INPUT; BINARY(4)
The absolute screen column for the new upper left corner of the window.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA3A2 E Window does not fit on screen.
CPFA3A4 E Specified window is not active.
CPFA3AA E Window handle incorrect.

Move Window by User (QsnMovWinUsr) API

Parameters

Required Parameter:			
1	Window handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Move Window by User (QsnMovWinUsr) API allows a window to be moved on the screen to a new location specified by the user. The API positions the cursor at the upper left corner of the window and prompts the user to move the cursor to the new position for the upper left corner. The prompt is displayed only if a message line has been defined. If the window can fit within the display at the location specified, it is moved to the new position. If a Change Window

Coordinates exit routine is specified on the window description, it is called after the window is successfully moved.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window to be moved.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA3A4 E Specified window is not active.
CPFA3AA E Window handle is incorrect.

Resize Window (QsnRszWin) API

Parameters

Required Parameter Group:

1	Window handle	Input	Binary(4)
2	Number of rows	Input	Binary(4)
3	Number of columns	Input	Binary(4)

Optional Parameter:

4	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Resize Window (QsnRszWin) API allows a window to be resized. If a Change Window Coordinates exit routine is specified on the window description, it is called after the window is successfully resized.

Required Parameter Group

Window handle
INPUT; BINARY(4)
A handle for the window to be resized.

Number of rows
INPUT; BINARY(4)
The new value for the number of rows in the window.

Number of columns
INPUT; BINARY(4)
The new value for the number of columns in the window.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3A2 E Window does not fit on screen.
CPFA3A4 E Specified window is not active.
CPFA3A5 E Window dimensions not within window limits.
CPFA3AA E Window handle incorrect.

Resize Window by User (QsnRszWinUsr) API

Parameters

Required Parameter:			
1	Window handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Resize Window by User (QsnRszWinUsr) API allows a window to be resized according to the cursor movements specified by the user. If the cursor is located on a border when this API is called, then that border can be moved. If the cursor is located on a border corner, the two sides that meet at that corner can be moved. If the user positions the cursor to a new row/column for the horizontal or vertical border, the border is moved to the new coordinate position and the window is resized accordingly. If the cursor is not on the border when the API is called, then the cursor is moved to the bottom right corner of the window and the user is prompted to move the cursor to the new position for the bottom right corner of the window. The prompt is displayed only if a message line has been defined.

A window can be made only as small (large) as the minimum (maximum) size allowed for the window. If the user moves the cursor such that the resulting window will be smaller (larger) than the minimum (maximum) size allowed, the resulting window will be the minimum (maximum) size. If a Change Window Coordinates exit routine is specified on the window description, this routine is called after the window is successfully resized. Typically, this API would be called after the user presses a particular function key.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window to be resized.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3A4 E Specified window is not active.
CPFA3AA E Window handle incorrect.

Retrieve Window Data (QsnRtvWinDta) API

Parameters

Required Parameter:			
1	Window handle	Input	Binary(4)
Optional Parameter Group:			
2	User data pointer	Output	PTR(SPP)
3	Error code	I/O	Char(*)
Returned Value:			
	User data pointer	Output	PTR(SPP)

The Retrieve Window Data (QsnRtvWinDta) API returns a pointer to the user data for the given window.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window for which the user data should be returned.

Optional Parameter Group

User data pointer
OUTPUT; PTR(SPP)
A window pointer to the user data, as specified on the window description, for the given window.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

User data pointer
OUTPUT; PTR(SPP)
This API returns the value for the user data pointer parameter, or the null pointer if an error occurs.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1F E Pointer parameter is not on a 16-byte boundary.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA3AA E Window handle incorrect.

Retrieve Window Description (QsnRtvWinD) API

Parameters

Required Parameter Group:

1	Window handle	Input	Binary(4)
2	Window description	Output	Char(*)
3	Length of window description	Input	Binary(4)

Optional Parameter:

4	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Retrieve Window Description (QsnRtvWinD) API retrieves a copy of the window description for the given window.

Required Parameter Group

Window handle
INPUT; BINARY(4)
A handle for the window for which the window description should be returned.

Window description
OUTPUT; CHAR(*)
The window description for the given window. The format of the data returned is shown in "Format of the Window Description Returned."

Length of window description
INPUT; BINARY(4)
The length of the window description parameter. If the length is larger than the size of the receiver variable, the results are not predictable. The minimum length is 8. The API returns as much information as it can fit in this length. If the available information is longer, it is truncated. If the available information is shorter, the unused output 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. To determine how much information the API could return if space were available, see the bytes available field.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Window Description Returned

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(*)	Window description

Field Descriptions

Bytes available. The number of bytes of data available to be returned.

Bytes returned. The number of bytes of data returned.

Window description. The format of the remaining data returned is shown in "Format of the Window Description" on page 20-3.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.
CPFA3AA E Window handle incorrect.

Set Window Services Attributes (QsnSetWinAtr) API

Parameters

Required Parameter Group:

1	Window services attributes description	Input	Char(*)
2	Length of window service attributes description	Input	Binary(4)

Optional Parameter:

3	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Set Window Services Attributes (QsnSetWinAtr) API sets the default attributes for the window services.

Required Parameter Group

Window services attributes
INPUT; CHAR(*)
Defines the attributes for the window services APIs. The format of the window services attributes description is shown in "Format of the Window Services Attribute Description."

Length of window services attributes
INPUT; BINARY(4)
The length of the window services attributes parameter.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Window Services Attribute Description

Offset			
Dec	Hex	Type	Field
0	0	CHAR(3)	Monochrome window display attributes
3	3	CHAR(3)	Color window display attributes

Field Descriptions

Color window display attributes. The window display attributes for a color display. The first character is the attribute for the window border when the window is not current, the second for when the window is current, and the third for the leading window attribute. All bytes can contain the same value. The special value X'00' indicates that no screen attribute is to be used for the given character. For each attribute field, both the monochrome and the color part must be either X'00' or a valid attribute. For example, it is incorrect to specify the second color attribute field as X'00' and the second monochrome attribute with a valid attribute.

The default values for these fields are X'20' for green, X'3A' for blue, and X'20' for green.

Monochrome window display attributes. The window display attributes for a monochrome display. The first character is the attribute for the window border when the window is not current, the second for when the window is current, and the third for the leading window attribute. All bytes may contain the same value. The special value X'00' indicates that no screen attribute is to be used for the given character. For each attribute field, both the monochrome and the color part must be either X'00' or a valid attribute. For example, it is incorrect to specify the second color attribute field as X'00' and the second monochrome attribute with a valid attribute.

The default values for these fields are X'20' for normal attribute, X'22' for high intensity, and X'20' for normal attribute, respectively.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.
CPFA3AC E Window services attributes description value is incorrect.

Chapter 21. Window I/O APIs

The majority of window I/O operations are performed through calls to the low-level services interfaces. If specified on the window description or through an explicit call to the Set Low-Level Environment Window Mode (QsnSetEnvWinMod) API, the low-level interfaces can operate in a relative mode, where operations such as Set Field (QsnSetFld) are performed relative to the current window. See "Set Low-Level Environment Window Mode (QsnSetEnvWinMod) API" on page 15-17 for details.

The APIs specific to window I/O operations are:

Clear Window (QsnClrWin) clears the window area.
Clear Window Message (QsnClrWinMsg) clears the message for a given window.
Display Window (QsnDspWin) draws the window border and clears the window area.
Put Window Message (QsnPutWinMsg) puts a message on the message line for a given window.

The detailed API descriptions are presented in alphabetical order.

Clear Window (QsnClrWin) API

Parameters

Required Parameter:			
1	Window handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Clear Window (QsnClrWin) API clears the window area for the given window. Any field definitions remain intact. Use the Clear Field Table (QsnClrFldTbl) API to remove field definitions.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window to be cleared.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3A4 E Specified window is not active.
CPFA3AA E Window handle incorrect.

Clear Window Message (QsnClrWinMsg) API

Parameters

Required Parameter:			
1	Window handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Clear Window Message (QsnClrWinMsg) API clears the window message for the given window. This API is valid only if the window has a message line specified for it.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window containing the message to be cleared.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter"

on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3A4 E Specified window is not active.
CPFA3A7 E Window does not have a message line.
CPFA3AA E Window handle incorrect.

Display Window (QsnDspWin) API

Parameters

Required Parameter:			
1	Window handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Display Window (QsnDspWin) API draws the window border for the current window and clears the window area. The QsnDspWin API does not make a window current. It simply redraws the window using the existing border attributes. For overlapping windows, use this API only for the current window. If a Draw Window exit routine is specified on the window description, this routine is called after the window is defined successfully and prior to actually drawing the window.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window to be drawn.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA3A4 E Specified window is not active.
CPFA3AA E Window handle incorrect.

Put Window Message (QsnPutWinMsg) API

Parameters

Required Parameter:			
1	Window handle	Input	Binary(4)
Optional Parameter Group:			
2	Message	Input	Char(*)
3	Message length	Input	Binary(4)
4	Lock keyboard	Input	Char(1)
5	Message identifier	Input	Char(7)
6	Qualified message file name	Input	Char(20)
7	Row	Input	Binary(4)
8	Column	Input	Binary(4)
9	Starting monochrome attribute	Input	Char(1)
10	Ending monochrome attribute	Input	Char(1)
11	Starting color attribute	Input	Char(1)
12	Ending color attribute	Input	Char(1)
13	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Put Window Message (QsnPutWinMsg) API places an error message on the message line for the given window. This API is valid only if the window has a message line specified for it.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window in which the message should be placed.

Optional Parameter Group

Message
INPUT; CHAR(*)
The message to be displayed. If the message does not fit within the window, it is truncated to fit. If the message length parameter is specified as nonzero, the message parameter is required. The message or the message ID parameter must be specified. If the message parameter is specified, the message ID parameter is ignored and no help key support is available for the message.

Message length
INPUT; BINARY(4)
The number of bytes of message data to be displayed.

Lock keyboard
INPUT; CHAR(1)
Indicates whether the keyboard should be locked or not. The possible values are:

0 Do not lock keyboard
1 Lock keyboard

Message identifier
INPUT; CHAR(7)
The identifying code for the predefined message to be displayed. The first level text is displayed. If the user moves the cursor to the message line and presses the Help key, the message `No help text available` is displayed. This parameter is required if the message parameter is omitted.

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 the message file resides. This parameter is required if the message parameter is omitted.

The format of this parameter is:

Bytes	Value
1-10	Message file name
11-20	Message file library. This can be an actual library name or one of the special values *CURLIB or *LIBL.

Row
INPUT; BINARY(4)
The relative window row at which to position the cursor when the message is displayed.

If both row and column are omitted or specified with a zero value, the cursor is not moved. Row and column must both be specified or omitted; one cannot be specified if the other is omitted.

Column
INPUT; BINARY(4)
The relative window column at which to position the cursor when the message is displayed.

Starting monochrome attribute
INPUT; CHAR(1)
The initial screen attribute for monochrome displays. If this parameter is omitted and monochrome attributes are to be used, no initial attribute is written to the display for the data.

The monochrome attribute and color attribute parameters consist of 2 bytes each: an initial and ending screen attribute to be used for a monochrome or a color display, respectively. One of these parameters will be selected based on the underlying display type, and the other will be discarded. Any of the attributes can be specified as a special value, X'00', indicating that no screen attribute should be written to the display. If the initial screen attribute is specified as an actual attribute, the data column, if specified, must be greater than or equal to 2. The initial screen attribute, if not X'00', will be written to the screen at the column previous to the first data character if row and column are specified, otherwise to the current display address. The ending screen attribute, if not X'00', will be written at the column directly after the last data character.

See "Screen Attribute Characters" on page 14-3 for a description of the screen attribute values.

Ending monochrome attribute
INPUT; CHAR(1)
The ending screen attribute for monochrome displays. If this parameter is omitted and monochrome attributes are to be used, no ending attribute is written to the display for the data.

Starting color attribute
INPUT; CHAR(1)
The initial screen attribute for color displays. If this parameter is omitted and color attributes are to be used, no initial attribute is written to the display for the data.
See "Screen Attribute Characters" on page 14-3 for a description of the screen attribute values.

Ending color attribute
INPUT; CHAR(1)
The ending screen attribute for color displays. If this parameter is omitted and color attributes are to be used, no ending attributes are written to the display for the 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. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA307 E Screen position &1, &2 outside of display or window area.
CPFA30D E Invalid screen attribute.
CPFA31E E Required parameter &1 omitted.
CPFA333 E Parameter &1 not positive integer value.
CPFA335 E Screen address parameter error.
CPFA3A4 E Specified window is not active.
CPFA3A7 E Window does not have a message line.
CPFA3A8 E Error occurred retrieving message text.
CPFA3AA E Window handle incorrect.
CPFA3AB E The value for parameter &1 must be 0 or 1.

Chapter 22. Window Manager Services APIs

The window manager services APIs manage multiple windows, support I/O to several active windows, and allow switching between windows. Much of the work of the window manager services is performed implicitly through the window builder routines.

Windows are managed on an activation-group basis. That is, all windows that were started within a given activation group will be managed as a unit. You can only switch to or end a window that was started within the current activation group. If windows need to be redrawn, only those windows within the current activation group will be redrawn. A window can be created in one activation group and started in another group. The activation group in which the window was started will be the group to manage the window.

The window manager services APIs are:

End a Window (QsnEndWin) ends an active, current window and removes it from the screen.
Retrieve Current Window (QsnRtvCurWin) returns the handle for the current window.
Set Current Window (QsnSetCurWin) makes the specified window current.
Start a Window (QsnStrWin) starts a window by displaying it and making it the current window.

The detailed API descriptions are presented in alphabetical order.

End a Window (QsnEndWin) API

Parameters

Required Parameter:			
1	Window handle	Input	Binary(4)
Optional Parameter Group:			
2	Restore screen	Input	Char(1)
3	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The End a Window (QsnEndWin) API ends a currently active window that was started with the Start a Window (QsnStrWin) API. The window is removed from the display on the screen and from the active window list. The data associated with the window is not deallocated.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window to be ended.

Optional Parameter Group

Restore screen
INPUT; CHAR(1)
Indicates if the underlying display image should be restored when the window is ended. This parameter is ignored if the underlying display image was not saved. This option should be used if the screen will be refreshed by another application and does not need to be refreshed when the window is removed. Performance can be improved by not restoring the display image. However, the saved screen may not be restored properly if it is not restored by another application.

The possible values are:

0 Do not restore the screen when the window is ended.
1 Restore the screen, if saved, when the window is ended. This is the default.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3A4 E Specified window is not active.
CPFA3AA E Window handle incorrect.
CPFA3AB E The value for parameter &1 must be 0 or 1.

Retrieve Current Window (QsnRtvCurWin) API

Parameters

Optional Parameter Group:

1	Current window handle	Output	Binary(4)
2	Error code	I/O	Char(*)

Returned Value:

	Current window handle	Output	Binary(4)

The Retrieve Current Window (QsnRtvCurWin) API returns the handle for the current window.

Optional Parameter Group

Current window handle
OUTPUT; BINARY(4)
The variable that contains the handle for the current window when the QsnRtvCurWin API has completed. If there is no current window, this parameter is set to 0.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Current window handle
OUTPUT; BINARY(4)
This API returns the value for the current window handle parameter. If there is no current window, this API returns 0.

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.

Set Current Window (QsnSetCurWin) API

Parameters

Required Parameter:

1	Window handle	Input	Binary(4)

Optional Parameter:

2	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Set Current Window (QsnSetCurWin) API makes the given window the current window. The QsnSetCurWin API draws the window with the current window border attribute, if specified. The Current Window exit routine, if specified on the window description, is called after the given window becomes current. The current window overlays all other windows on the display screen.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window that will become current.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA3A4 E Specified window is not active.
CPFA3AA E Window handle incorrect.

Start a Window (QsnStrWin) API

Parameters

Required Parameter:

1	Window handle	Input	Binary(4)

Optional Parameter Group:

2	Save screen	Input	Char(1)
3	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Start a Window (QsnStrWin) API starts a window created with the Create a Window (QsnCrtWin) API. This causes the window to be displayed on the screen and added to the active window list. If specified, the Draw Window exit routine is called immediately before the window is drawn.

Required Parameter

Window handle
INPUT; BINARY(4)
A handle for the window to be started.

Optional Parameter Group

Save screen
INPUT; CHAR(1)
Indicates if the underlying display image should be saved prior to drawing the window. This option should be used only if the window will not be moved or resized over an existing display image. Performance can be improved by not saving the display image. However, doing this limits the overlapping nature of the window. If an attempt is made to move or resize a window for which the display image was not saved, the screen is cleared and all windows are redrawn prior to moving the window.

The possible values for this parameter are:

0 Do not save the underlying display image when the window is started.
1 Save the underlying display image when the window is started. This is the default.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA318 E Error calling exit routine.
CPFA31E E Required parameter &1 omitted.
CPFA3AA E Window handle incorrect.
CPFA3AB E The value for parameter &1 must be 0 or 1.

Performance Considerations

You can improve the performance of window operations by doing the following:

- Do not save or restore the underlying screen image when a window is started or ended with the Start a Window (QsnStrWin) or End a Window (QsnEndWin) API, respectively. See pages 22-2 and 22-1.
- For non-GUI windows, use the same color for current and noncurrent boundaries.
- Use a display station attached to a control unit that supports an enhanced interface for a nonprogrammable work station, even if you are not using GUI windows.
- Use GUI window support when the underlying control unit supports this.

Creating/Manipulating Windows Example

The sample program in Figure 22-1 on page 22-4 shows how to create and manipulate several windows with exit routines. The program creates three windows– Window 1, Window 2, and Window 3. Each time Enter is pressed, the next window is made current; in which case, the Draw Window exit routine for that window is called. If the user presses F4=Move or F5=Resize, the current window is moved or resized and the Draw Window exit routine is called again. The resulting screen output is shown in Figure 22-2 on page 22-5.

Creating/Manipulating Windows Example

```
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "qsnll.h"
#include "qsnwin.h"
#include "qsnerr.h"


void GenericDraw(const Qsn_Cmd_Buf_T *cbuf, const Qsn_Win_T *win)
{
    char *msg1 = "F3: quit F4: move  F5: resize";
    char *msg2 = "text no attribute";

    QsnWrtDta(msg2, strlen(msg2), 0, 2, 1, QSN_NO_SA, QSN_NO_SA,
              QSN_NO_SA, QSN_NO_SA, *cbuf, *win, NULL);
    QsnWrtDta(msg1, strlen(msg1), 0, -1, 1, QSN_SA_HI, QSN_SA_NORM,
              QSN_SA_RED, QSN_SA_NORM, *cbuf, *win, NULL);
}

void Draw1(const Qsn_Win_T *win, const Qsn_Cmd_Buf_T *cbuf)
{
    char *txt = "window 1 (ul/blue)";

    GenericDraw(cbuf, win);
    QsnWrtDta(txt, strlen(txt), 0, 5, 5, QSN_SA_UL, QSN_SA_NORM,
              QSN_SA_BLU, QSN_SA_NORM, *cbuf, *win, NULL);
}

void Draw2(const Qsn_Win_T *win, const Qsn_Cmd_Buf_T *cbuf)
{
    char *txt = "window 2 (ul/red)";

    GenericDraw(cbuf, win);
    QsnWrtDta(txt, strlen(txt), 0, 5, 5, QSN_SA_UL, QSN_SA_NORM,
              QSN_SA_RED, QSN_SA_NORM, *cbuf, *win, NULL);
}

void Draw3(const Qsn_Win_T *win, const Qsn_Cmd_Buf_T *cbuf)
{
    char *txt = "window 3 (ul/pink)";

    GenericDraw(cbuf, win);
    QsnWrtDta(txt, strlen(txt), 0, 5, 5, QSN_SA_UL, QSN_SA_NORM,
              QSN_SA_PNK, QSN_SA_NORM, *cbuf, *win, NULL);
}
```

Figure 22-1 (Part 1 of 2). Creating and Manipulating Windows

```
int main (void) {
    int i;
    char textff100";
    Qsn_Win_T win1, win2, win3, cur;
    Qsn_Win_Desc_T  win_desc;
    Qsn_Win_Ext_Inf_T ext = { NULL, NULL, NULL, NULL, NULL, NULL };
    Q_Bin4          win_desc_length = sizeof(win_desc);
    char aid;

    QsnInzWinD(&win_desc, win_desc_length, NULL);
    win_desc.GUI_support = '0';

    /* define and start window 1 */
    win_desc.top_row = 3;
    win_desc.left_col = 5;
    win_desc.num_rows = 13;
    win_desc.num_cols = 40;
    ext.draw_fp = Draw1;
    win1 = QsnCrtWin(&win_desc, win_desc_length, &ext, sizeof(ext),
                     '1', NULL, 0, NULL, NULL);
    QsnGetAID(NULL, 0, NULL);

    /* define and start window 2 */
    win_desc.top_row = 10;
    win_desc.left_col = 10;
    win_desc.num_rows = 10;
    win_desc.num_cols = 30;
    ext.draw_fp = Draw2;
    win2 = QsnCrtWin(&win_desc, win_desc_length, &ext, sizeof(ext),
                     '1', NULL, 0, NULL, NULL);
    QsnGetAID(NULL, 0, NULL);

    /* define and start window 3 */
    win_desc.top_row = 5;
    win_desc.left_col = 20;
    win_desc.num_rows = 15;
    win_desc.num_cols = 50;
    ext.draw_fp = Draw3;
    win3 = QsnCrtWin(&win_desc, win_desc_length, &ext, sizeof(ext),
                     '1', NULL, 0, NULL, NULL);
    cur = win3;

    for ( ;; ) {
        if (( (aid=QsnGetAID(NULL, 0, NULL)) == QSN_F3))
            break;
        else if (aid == QSN_F4)
            QsnMovWinUsr(cur, NULL);
        else if (aid == QSN_F5)
            QsnRszWinUsr(cur, NULL);
        else {
            /* switch current window to next window */
            if (cur == win1) {
                QsnSetCurWin(win2, NULL);
                cur = win2;
            } else if (cur == win2) {
                QsnSetCurWin(win3, NULL);
                cur = win3;
            } else {
                QsnSetCurWin(win1, NULL);
                cur = win1;
            }
        }
    }

}
```

Figure 22-1 (Part 2 of 2). Creating and Manipulating Windows

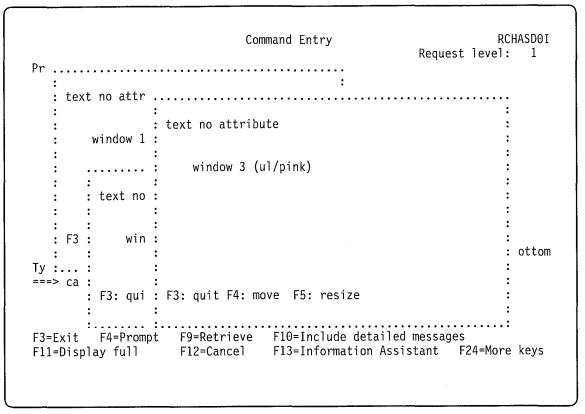


Figure 22-2. Display Screen

Chapter 23. Introduction to the Session Services APIs

The session services APIs provide a general scrolling I/O interface. They can be used to build a standard input-line scrolling interface or an interface that has an output-only scroll area (called a **scroller**) in a window. Sessions are special cases of windows as supported by the window services. A session is defined using a session, a window, and a low-level environment description. The window and low-level environment descriptions are the same as those used to define a window directly with the window services APIs. The session description defines the structure of the session. The structure includes the coordinates of the scrolling portion, the length of the input line, the amount to roll by, and so on. A session is implemented as a window, where the window user data pointer describes the session itself. Thus, a session can be manipulated through the window and low-level interfaces by passing the session handler or through the session interfaces. This implementation is similar to the concept of inheritance in object-oriented programming languages.

Sessions are similar in concept to subfiles and can be used for any application that requires a scrolling line interface. The session services APIs are divided into the following functional groups:

- **Session manipulation and query APIs** allow you to create, query, and manipulate sessions.
- **Session I/O APIs** allow you to perform input and output operations to sessions.

Session Details

Figure 23-1 shows the default attributes provided by the DSM session description for a session.

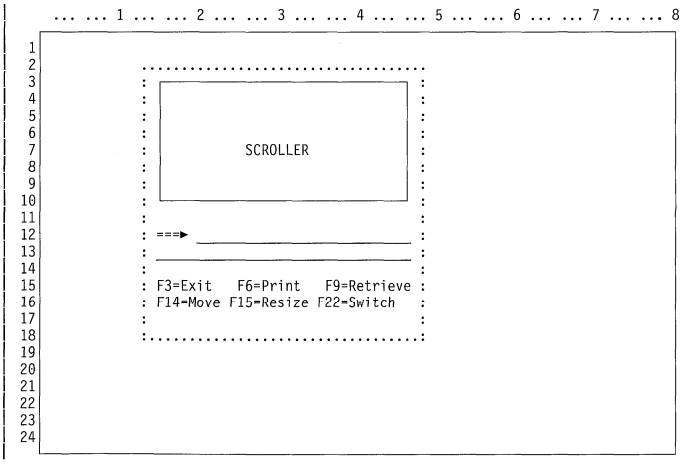


Figure 23-1. Session Attributes

The main component of a session is the scrollable area, or **scroller**, where output data can be displayed for the session. A session may or may not have a data input line, depending on the application. A session that uses the default attributes has an input line underneath the scroller. You can allow the size and location of the session attributes to default based on the window size, or you can specify these explicitly. Up to two lines of command key descriptions can appear below the scroller and can be managed by a session. For details on the session description see "Create a Session (QsnCrtSsn) API" on page 24-2.

When a window containing a session is moved or resized, the scroller and any automatically defined fields are redrawn to reflect the new window positions and size. If any additional items have been added to the session through the low-level interface APIs, you must supply an exit program that will reposition such items explicitly. See "Create a Session (QsnCrtSsn) API" on page 24-2 for details on the exit program.

Line Mode and Character Mode I/O

Session I/O can be performed in a line mode or a character mode basis. In line mode, each call to the line-specific interfaces operates on a complete line, either on input or output. In character mode, I/O is performed a character at a time. This means that multiple I/O operations can be issued to operate on the current line. For example, an output operation could output several characters. Then a backspace operation could be followed by input from the current cursor position. (All input operations are still performed in block mode, where the input is not available until an AID-generating key has been pressed.)

Line mode output is performed using the Write Line to Scroller (QsnWrtSclLin) API. This API writes a line of data to the next session line and sets the active position (see "Active Position" on page 23-2) to the start of the next line after the added line. For character output, the Write Characters to Scroller (QsnWrtSclChr) API is used. This API outputs a string of characters starting at the active position. After this operation completes, the active position is one position past the last character written, or it is the position specified by a control character sequence if this appears at the end of the data.

Command Key Action Routines

Part of the session description is an array of command key actions. Each action is an exit routine that is specified as a function pointer. When a command key is pressed during a QsnReadSsnDta operation, if an action has been specified, the appropriate exit routine is called. Otherwise, an `Invalid key pressed` error message will be issued. DSM provides a group of functions that can be called, or user-defined exit-routines can be specified. The action routines are specified as an array of 24 function pointers in the session description. (See "Create a Session (QsnCrtSsn) API" on page 24-2 for

details.) The default values for the action routines DSM calls are:

Cmd Key	Action Routine
1	
2	
3	
4	
5	
6	Print Scroller Data (QsnPrtScl)
7	Roll Scroller Down (QsnRollSclDown)
8	Roll Scroller Up (QsnRollSclUp)
9	Retrieve Session Input Line to Command Line (QsnRtvSsnLin)
10	
11	Toggle Line Wrap/Truncate Mode (QsnTglSclWrp)
12	
13	
14	Move Window by User (QsnMovWinUsr)
15	Resize Window by User (QsnRszWinUsr)
16	
17	Display Scroller Top (QsnDspSclT)
18	Display Scroller Bottom (QsnDspSclB)
19	Shift Scroller Left (QsnShfSclL)
20	Shift Scroller Right (QsnShfSclR)
21	Display Command Line Window (direct mapping to QUSCMDLN API)
22	
23	
24	

The default action routines for command keys 7, 8, 19, and 20 (QsnRollSclDown, QsnRollSclUp, QsnShfSclL, and QsnShfSclR, respectively) will pass any numeric input to the API before the command key is pressed. For example, to shift the scroller to the right by 10 columns, the value 10 could be entered at the input line prior to pressing command key 20. Non-numeric input is ignored.

When a user-defined action routine is called, it is passed the following information:

Information Passed to the Action Routine

1	Session handle	Input	Binary(4)
2	Input buffer	Input	Binary(4)
3	Returned action	Output	Char(1)

When the specified command key is pressed, the action routine for the command key is called. If you change the default values to have a command key call a different DSM API, you cannot specify the API directly because the action routine is passed specific parameters. You must define an action routine that can accept the action routine parameters and then call the desired DSM API with the appropriate parameters. You can define a generic action routine that is specified for each key you want to define, and in that action routine query the input buffer to determine the command key pressed and the appropriate action to take.

Action Routine Parameters

Session handle
INPUT; BINARY(4)
The session currently active. If the action routine causes the active session to change, this variable will be changed to reflect the new session.

Input buffer
INPUT; BINARY(4)
The input buffer containing the results of the input operations that caused this exit routine to be called. The input buffer can be queried using the low-level interface routines.

Returned action
OUTPUT; CHAR(1)
The variable containing the flag indicating if, following a successful return from this exit routine, control returns to the caller of the Read Data from Session (QsnReadSsnDta) API or if QsnReadSsnDta handles the next input operation. If an error occurs in the exit routine, control always returns to the caller. The possible values are:

0 QsnReadSsnDta continues to handle the next input operation. Control does not return to the caller.
1 QsnReadSsnDta returns control to the caller. The output parameters for QsnReadSsnDta are filled in appropriately.

Active Position

The active position in the scroller is the point at which data will be written for character mode operations. The active position is affected by output operations to the scroller, including the writing of data that contains EBCDIC display control character sequences.

EBCDIC Display Control Characters

The data written to the scroller may contain display control characters consisting of single byte EBCDIC values. If specified on the session description (see "Create a Session (QsnCrtSsn) API" on page 24-2), the APIs for writing data to the scroller will check for and interpret such control characters. Each control character recognized in the output data is replaced by a call to a DSM API or internal routine that will perform the appropriate function. Figure 23-2 shows the control characters that are recognized and the APIs that are called, where applicable.

Figure 23-2. EBCDIC Display Control Characters

Character	Hex Value	Interpretation
HT	05	QsnSclTab

Figure 23-2. EBCDIC Display Control Characters

Character	Hex Value	Interpretation
VT	0B	QsnSclLF
FF	0C	QsnSclFF
CR	0D	QsnSclCR
NL	15	QsnSclNL
BS	16	QsnSclBS
BEL	2F	QsnBeep

DBCS Considerations

If the low-level environment description (see "Format of the Low-Level Environment Description" on page 15-4) for the session specifies DBCS support, the session services will check for and handle DBCS data. DBCS data must be enclosed by shift control (SO/SI) characters. The DBCS support field determines the type of the input field defined for the session, but does not affect the checking done for session output data other than to indicate that DBCS data may be present. The scroller does not display data using extended NLS attributes, regardless of the underlying display device support.

If DBCS support is specified, the wrap indication for the session description must always be set to 1. Also, line retrieval is not supported for DBCS sessions.

Chapter 24. Session Manipulation and Query APIs

The session manipulation and query APIs are:

- **Change Session** (QsnChgSsn) changes the description for a session.
- **Clear Scroller** (QsnClrScl) clears the scroller data.
- **Create a Session** (QsnCrtSsn) creates a session for subsequent session I/O operations.
- **Display Scroller Bottom** (QsnDspSclB) shows the last line of scroller data.
- **Display Scroller Top** (QsnDspSclT) shows the first line of scroller data.
- **Initialize Session Description** (QsnInzSsnD) initializes a session description with default values.
- **Query If Scroller in Line Wrap Mode** (QsnQrySclWrp) queries if line wrap mode is on or off.
- **Retrieve Number of Columns to Shift Scroller** (QsnRtvSclNumShf) returns number of columns to shift scroller by.
- **Retrieve Number of Rows to Roll Scroller** (QsnRtvSclNumRoll) returns the number of rows to roll scroller by.
- **Retrieve Session Data** (QsnRtvSsnDta) returns a pointer to the user data for a session.
- **Retrieve Session Description** (QsnRtvSsnD) retrieves a copy of the description for a session.
- **Roll Scroller Down** (QsnRollSclDown) rolls the scroller down.
- **Roll Scroller Up** (QsnRollSclUp) rolls the scroller up.
- **Shift Scroller Left** (QsnShfSclL) shifts the scroller to the left.
- **Shift Scroller Right** (QsnShfSclR) shifts the scroller to the right.
- **Toggle Line Wrap/Truncate Mode** (QsnTglSclWrp) toggles the session between line wrap and truncation mode.

The detailed API descriptions are presented in alphabetical order.

Change Session (QsnChgSsn) API

Parameters

Required Parameter Group:

1	Session handle	Input	Binary(4)
2	Session description	Input	CHAR(*)
3	Length of session description	Input	Binary(4)

Optional Parameter:

4	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Change Session (QsnChgSsn) API changes the session description for the given session. If the session contains DBCS data, the input line or the number of columns in the scroller cannot be changed. If the session is currently displayed, it will be redrawn to reflect any changes.

Required Parameter Group

Session handle
INPUT; BINARY(4)
A handle for the session for which the session description is to be changed.

Session description
INPUT; CHAR(*)
The new session description for the given session. The format of the session description is shown in "Format of the Session Description" on page 24-3.

Length of session description
INPUT; BINARY(4)
The length of the session description parameter.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA314 E Memory allocation error.
CPFA318 E Error calling exit routine.
CPFA340 E Operation not supported with double-byte data.
CPFA31E E Required parameter &1 omitted.
CPFA3D1 E Scroller printed.
CPFA3D6 E Session handle is incorrect.

Clear Scroller (QsnClrScl) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter Group:			
2	Resize indication	Input	Char(1)
3	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Clear Scroller (QsnClrScl) API clears the scroller data associated with a session and optionally resizes the scroller buffer.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be cleared. All data in the scroller will be cleared.

Optional Parameter Group

Resize indication
Input; CHAR(1)
Whether the scroller buffer should be resized when it is cleared.

0 Maintain current buffer size and data. This is the default.
1 Resize the buffer to the initial size given on the session description.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA314 E Memory allocation error.
CPFA31E E Required parameter &1 omitted.
CPFA3D6 E Session handle is incorrect.
CPFA3AB E The value for parameter &1 must be 0 or 1.

Create a Session (QsnCrtSsn) API

Parameters

Required Parameter Group:			
1	Session description	Input	Char(*)
2	Length of session description	Input	Binary(4)
Optional Parameter Group:			
3	User extension information	Input	Char(*)
4	Length of user extension information	Input	Binary(4)
5	Start session flag	Input	Char(1)
6	Window description	Input	Char(*)
7	Length of window description	Input	Binary(4)
8	Low-level environment description	Input	Char(*)
9	Length of low-level environment description	Input	Binary(4)
10	Session handle	Output	Binary(4)
11	Error code	I/O	Char(*)
Returned Value:			
	Session handle	Output	Binary(4)

The Create a Session (QsnCrtSsn) API creates a session and returns a handle for the created session. The session must be deleted using the Delete Low-Level Environment (QsnDltEnv) API.

Required Parameter Group

Session description
INPUT; CHAR(*)
The defined attributes of the session to be created. It must be declared aligned on a 16-byte boundary. The format of the session description is shown in "Format of the Session Description" on page 24-3.

Length of session description
INPUT; BINARY(4)
The length of the session description parameter.

Optional Parameter Group

User extension information
INPUT; CHAR(*)
Information that is used to associate data and exit routines with the session. This parameter is required if the user extension information length parameter is supplied. This essentially enables the object-oriented programming concept of inheritance, allowing the session to be

extended in a natural way. The user extension data cannot be changed once the session has been created. The format of this parameter is shown in the section "Format of the Session User Extension Data" on page 24-5.

Length of user extension information
INPUT; BINARY(4)
The length of the user extension information parameter.

Start session flag
INPUT; CHAR(1)
Whether or not the session should be displayed on screen when it is created. The possible values are:

0 Do not display the session on the screen when it is created. If you specify this value, you must use the Start a Window (QsnStrWin) API to display the session.
1 Display the session on the screen when it is created. This is the default.

Window description
INPUT; CHAR(*)
The defined attributes for the window containing the session. This parameter is required if the window description length parameter is supplied. The format of the window description is shown in "Format of the Window Description" on page 20-3. If this parameter is omitted, a window will be created with default values.

Length of window description
INPUT; BINARY(4)
The length of the window description parameter.

Low-level environment description
INPUT; CHAR(*)
The low-level environment description that defines the operating environment for low-level operations used to create and manipulate the windows. This parameter is required if the low-level environment description length parameter is supplied. The format of the low-level environment description is shown in "Format of the Low-Level Environment Description" on page 15-4. If this parameter is omitted, a low-level environment will be created with default values.

Length of low-level environment description
INPUT; BINARY(4)
The length of the low-level environment description parameter.

Session handle
OUTPUT; BIN(4)
The variable containing a handle for the created session after the QsnCrtSsn API has completed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Session handle
OUTPUT; BINARY(4)
This API returns the value for the session handle parameter.

Format of the Session Description

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	PTR(PP) [24]	Array of command key actions
384	180	BINARY(4)	Top row of scroller
388	184	BINARY(4)	Left column of scroller
392	188	BINARY(4)	Number of rows in scroller
396	18C	BINARY(4)	Number of columns in scroller
400	190	BINARY(4)	Default number of rows to roll scroller by
404	194	BINARY(4)	Default number of columns to shift scroller by
408	198	BINARY(4)	Scroller buffer initial size
412	19C	BINARY(4)	Scroller buffer maximum size
416	1A0	BINARY(4)	Scroller buffer increment
420	1A4	BINARY(4)	Number of rows for input line
424	1A8	CHAR(1)	Reserved
425	1A9	CHAR(1)	Wrap indication
426	1AA	CHAR(1)	Reserved
427	1AB	CHAR(1)	Display control characters indication
428	1AC	CHAR(1)	Echo session input
429	1AD	CHAR(1)	Scroller line display
430	1AE	CHAR(1)	Show scroller characters
431	1AF	CHAR(1)	Show command key descriptions
432	1B0	CHAR(1)	Command key attribute for a monochrome display
433	1B1	CHAR(1)	Command key attribute for a color display
434	1B2	CHAR(2)	Reserved
436	1B4	BINARY(4)	Offset to input line prompt
440	1B8	BINARY(4)	Length of input line prompt
444	1BC	BINARY(4)	Offset to command key description line 1

Offset		Type	Field
Dec	Hex		
448	1C0	BINARY(4)	Length of command key description line 1
452	1C4	BINARY(4)	Offset to command key description line 2
456	1C8	BINARY(4)	Length of command key description line 2
460	1CC	BINARY(4)	Reserved.
*	*	CHAR(*)	Input line prompt
*	*	CHAR(*)	Command key description line 1
*	*	CHAR(*)	Command key description line 2

Field Descriptions

In the following descriptions, the default value refers to the value set by the Initialize Session Description (QsnInzSsnD) API.

Array of command key actions. An array of 24 function pointers, each corresponding to the action to be performed when the associated command key is pressed. An element that is specified as a null pointer indicates that the command key is invalid. An element can also be set to the dummy routine QsnSameAction, in which case the current action routine for that key is used. If a command key action is set to the dummy routine QsnDefaultAction, then the default action for that key is used. The defaults for command key actions and the parameters passed to the action routines are described in "Command Key Action Routines" on page 23-1. The procedures are exported as part of the service program that contains the DSM session services.

Command key attribute for a color display. The default value is X'28' for red.

Command key attribute for a monochrome display. The default value is X'00' for normal attribute.

Command key description line 1. The text string for the first line of command key descriptions.

Command key description line 2. The text string for the second line of command key descriptions.

Default number of columns to shift scroller by. The default number of columns to shift scroller by for the Shift Scroller Left (QsnShfSclL) and Shift Scroller Right (QsnShfSclR) APIs. This value must be a positive integer value. If 0 is specified, the default is the number of columns in the scroller less two (two scroller columns are reserved for the prefix area).

Default number of rows to roll scroller by. The default number of rows to roll scroller by for the Roll Scroller Up (QsnRollSclUp) and Roll Scroller Down (QsnRollSclDown) APIs. This value must be a positive integer value. If 0 is specified, the default is the number of rows in the scroller.

Display control characters indication. Specifies whether or not scroller lines contain EBCDIC display control characters. If the data contains such control characters and this is not indicated, unexpected results can occur. (See "EBCDIC Display Control Characters" on page 23-2 for details of the control characters supported and their interpretation.) The possible values are:

0 Scroller lines do not contain EBCDIC display control characters. This is the default.
1 Scroller lines contain EBCDIC display control characters.

Echo session input. Specifies whether lines entered at the session command line are to be echoed to the scroller. The possible values are:

0 Do not echo session input lines to the scroller.
1 Echo session input lines to the scroller. This is the default.

Input line prompt. The text string for the input line prompt.

Left column of scroller. This position is relative to the left of the window which is column 1. The default is 1.

Length of command key description line 1. This value must not exceed the maximum number of columns in the window. The default value is 0. No space is used in the session for this line. If the description line cannot be displayed completely within the window, it is truncated to fit.

Length of command key description line 2. This value must not exceed the maximum number of columns in the window. The default value is 0. No space is used in the session for this line. If the description cannot be displayed completely in the window, it is truncated to fit.

Length of input line prompt. This value must not exceed the maximum number of columns in the window. A value of 0 specifies that there is no prompt. The default value is -1 and corresponds to the default input line prompt ===>.

If the input line cannot be displayed completely within the window, it is truncated to fit. The input line will continue on the next window line.

Number of columns in scroller. This value must be a positive integer no greater than the number of columns in the session window. If 0 is specified, the default is the number of columns remaining in the window from the left column of the scroller. This value includes the 2 bytes used for the prefix area to the left of the scroller input line.

Number of rows for input line. The input line starts in the row specified by the formula: last window row less the number of rows required for input line less the number of rows required for the function key descriptions. The input line will start one column past the end of the input line prompt. If there is no input line prompt, then the input line starts one byte to the right of the leftmost usable column of

the window. If this value is 0, then no input line is created. The default is 1.

Number of rows in scroller. This value must be a positive integer no greater than the number of rows in the session window. If 0 is specified, the default is the number of rows remaining in the window from the top row of the scroller.

Offset to command key description line 1. The offset from the beginning of the structure to the start of the command key description line 1. This field is ignored if the length of command key description line 1 field specifies no command key description.

Offset to command key description line 2. The offset from the beginning of the structure to the start of the command key description line 2. This field is ignored if the length of command key description line 2 field specifies no command key description.

Offset to input line prompt. The offset from the beginning of the structure to the start of the input line prompt. This field is ignored if the length of input line prompt field specifies no prompt or the default prompt.

Reserved. This field must be set to X'00'.

Scroller buffer increment. Specifies, in bytes, the amount to increment the scroller buffer size by when the buffer is full and the buffer-full action is to increment the buffer size. The default value is 2000 bytes.

If the scroller buffer cannot be incremented because of insufficient resources, data at the beginning of the scroller will be removed to create space for the new data.

Scroller buffer initial size. The initial buffer size, in number of bytes, that will be allocated for storing the session scroller lines. The default value is 2000 bytes.

Scroller buffer maximum size. The maximum buffer size, in bytes, that will be allocated for storing the session scroller lines. The default value is 0, indicating no maximum size.

Show command key descriptions. Whether or not the function key description lines are to be shown. The possible values are:

0 Do not show function key descriptions.
1 Show function key descriptions. This is the default.

Show scroller characters. Whether or not characters written to the scroller in character mode are shown immediately on the screen. You can use the scroller line and character display options together to specify that groups of characters are not displayed immediately, but each complete line is. The possible values are:

0 Do not show characters on the screen as they are written. Use the Display Scroller Bottom (QsnDspSclB) API to display the scroller data on the screen. This is the default.
1 Show scroller characters on the screen as they are written.

Show scroller lines. Whether or not lines written to the scroller are shown immediately on the screen. The possible values are:

0 Do not show scroller lines on the screen as they are written. Use the Display Scroller Bottom (QsnDspSclB) API to display the scroller lines on the screen. This is the default.
1 Show scroller lines on the screen as they are written.

Top row of scroller. This position is relative to the top of the window, which is row 1. The default is 1.

Wrap indication. How to handle lines that do not fit within the session window. Possible values are:

0 Truncate lines that do not fit.
1 Wrap lines to the next line. This is the default. A value of 1 must be specified if the session contains DBCS data.

Format of the Session User Extension Data

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	PTR(SPP)	User data associated with the session
16	16	PTR(PP)	Exit routine to call when the session is changed
32	20	PTR(PP)	Exit routine to call when window is deleted
48	30	PTR(PP)	Exit routine to call when window coordinates are changed
64	40	PTR(PP)	Exit routine to call when window is drawn
80	50	PTR(PP)	Exit routine to call when this window made current

Field Descriptions

Exit routine to call when session changed. The exit routine to call when a session is changed using the Change Session (QsnChgSsn) API.

Exit routine to call when window coordinates changed. The exit routine to call when the window coordinates are changed using the Move Window (QsnMovWin), Move Window by User (QsnMovWinUsr), Resize Window (QsnRszWin), or Resize Window by User (QsnRszWinUsr), APIs.

Exit routine to call when window deleted. The exit routine to call when a window is deleted using the Delete Low-Level Environment (QsnDltEnv) API.

Exit routine to call when window drawn. The exit routine to call when a window is drawn using the Display Window (QsnDspWin) API.

Exit routine to call when window made current. The exit routine to call when this window is made current using the Set Current Window (QsnSetCurWin) API.

User data associated with the session. This is a pointer to any data that the user wants to associate with this session.

Session Exit Routines

Exit routines are user-supplied functions with a defined interface. The routines are called from certain APIs and allow the programmer to attach additional function to those APIs. For instance, if fields have been set up in a window, a Change Coordinates exit routine could be supplied to move the fields if the window is moved.

Change Session Exit Routine: This exit routine, if specified on the user extension information, is called when the session is changed. The following parameter is passed to the exit routine:

Parameter Passed to Exit Routine

1	Session handle	Input	Binary(4)

Change Session Exit Routine Parameter

Session handle
INPUT; BINARY(4)
The session that was changed.

Delete Session Exit Routine: This exit routine, if specified on the user extension information, is called when the session is deleted. The following parameter is passed to the exit routine:

Parameter Passed to Exit Routine

1	Session handle	Input	Binary(4)

Delete Session Exit Routine Parameter

Session handle
INPUT; BINARY(4)
The session that was deleted.

Change Session Coordinates Exit Routine: This exit routine, if specified on the user extension information, is called when the move or resize APIs are called, after the session has been successfully moved or resized. The following parameters are passed to the exit routine:

Parameters Passed to Exit Routine

1	Session handle	Input	Binary(4)
2	Top border offset	Input	Binary(4)
3	Left border offset	Input	Binary(4)
4	Bottom border offset	Input	Binary(4)
5	Right border offset	Input	Binary(4)

Change Session Coordinates Exit Routine Parameters

Session handle
INPUT; BINARY(4)
The session for which the coordinates were changed.

Top border offset
INPUT; BINARY(4)
The offset, in screen rows, from the previous top session border to the current top session border (after the session coordinates have been changed). It can be positive, negative, or zero, depending on how the top session border was changed. For example, if the top border was moved down two rows, this value would be 2; if it was moved up 4 rows, this value would be -4; if the top row was not changed, this value would be 0.

Left border offset
INPUT; BINARY(4)
The offset, in screen columns, from the previous left session border to the current left session border (after the session coordinates have been changed). It can be positive, negative, or zero, depending on how the left session border was changed. For example, if the left border was moved two columns to the right, this value would be 2; if it was moved 4 columns to the left, this value would be -4, and if the left column was not changed, this value would be 0.

Bottom border offset
INPUT; BINARY(4)
The offset, in screen rows, from the previous bottom session border to the current bottom session border (after the session coordinates have been changed). It can be positive, negative, or zero, depending on how the bottom session border was changed. For example, if the border was moved down two rows, this value would be 2; if it was moved up 4 rows, this value would be -4; if the bottom row was not changed, this value would be 0.

Right border offset
INPUT; BINARY(4)
The offset, in screen columns, from the previous right session border to the current right session border (after the session coordinates have been changed). It can be positive, negative, or zero, depending on how the right session border was changed. For example, if the right border was moved two columns to the right, this value would be 2; if it was moved 4 columns to the left, this value would be -4; if the right column was not changed, this value would be 0.

Draw Session Exit Routine: This exit routine, if specified on the user extension information, is called when the Display Window (QsnDspWin) API is called, before the session is drawn. The following parameters are passed to the exit routine:

Parameters Passed to Exit Routine

1	Session handle	Input	Binary(4)
2	Command buffer	Input	Binary(4)

Draw Session Exit Routine Parameters

Session handle
INPUT; BINARY(4)
The session to be drawn.

Command buffer
INPUT; BINARY(4)
The command buffer used to store the commands that re-create the window contents. The contents of the command buffer are written to the screen along with the window border. This allows the window and its contents to be redrawn in a single I/O operation.

Current Session Exit Routine: This exit routine, if specified on the user extension information, is called when the session is made current through the Set Current Window (QsnSetCurWin) API. The following parameter is passed to the exit routine:

Parameter Passed to Exit Routine

1	Session handle	Input	Binary(4)

Current Session Exit Routine Parameter

Session handle
INPUT; BINARY(4)
A handle for the session that is made current.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA314 E Memory allocation error.
CPFA318 E Error calling exit routine.
CPFA327 E Low-level environment description value incorrect.
CPFA31E E Required parameter &1 omitted.
CPFA3A1 E Window description value is incorrect.
CPFA3AB E The value for parameter &1 must be 0 or 1.
CPFA3D1 E Scroller printed.

Additional errors may be generated by this API. They are listed under the applicable API as follows:

Error Category/ API	Page Reference
Environment description/ QsnCrtEnv	15-6
Window description/ QsnCrtWin	20-1

Display Scroller Bottom (QsnDspScIB) API

Parameters

Required Parameter:

1	Session handle	Input	Binary(4)

Optional Parameter:

2	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Display Scroller Bottom (QsnDspScIB) API positions the scroller at the last line in the scroller area. As many lines preceding the last line as can fit in the scroller area are displayed as well.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be rolled.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3D6 E Session handle is incorrect.

Display Scroller Top (QsnDspScIT) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Display Scroller Top (QsnDspSclT) API positions the scroller at the first line in the scroller area. As many lines following the first line as can fit in the scroller area are displayed as well.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be rolled.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3D6 E Session handle is incorrect.

Initialize Session Description (QsnInzSsnD) API

Parameters

Required Parameter Group:			
1	Session description	Output	Char(*)
2	Length of session description	Input	Binary(4)
Optional Parameter:			
3	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Initialize Session Description (QsnInzSsnD) API initializes a session description with default values. Unless otherwise specified in the session description (see "Format of the Session Description" on page 24-3), pointer fields are set to the null pointer, numeric fields to 0, character flag fields to 0, and other character fields to blanks. For example, the default value for the wrap indication is 1, so this field will be set to 1.

Required Parameter Group

Session description
OUTPUT; CHAR(*)
The session description to be initialized

Length of session description
INPUT; BINARY(4)
The length of the session description parameter.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPF3C1D E Length specified in parameter &1 not valid.

Query If Scroller in Line Wrap Mode (QsnQrySclWrp) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter Group:			
2	Wrap indication	Output	Char(1)
3	Error code	I/O	Char(*)
Returned Value:			
	Wrap indication	Output	Binary(4)

The Query If Scroller in Line Wrap Mode (QsnQrySclWrp) API queries if line wrap mode is set on or off for the given session.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be queried.

Optional Parameter Group

Wrap indication
OUTPUT; CHAR(1)
Whether line wrap mode is on or off. The possible values are:

0 Line wrap mode is off.
1 Line wrap mode is on.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Wrap indication
OUTPUT; BINARY(4)
This API returns the value for the wrap indication parameter if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3D6 E Session handle is incorrect.

Retrieve Number of Columns to Shift Scroller (QsnRtvSclNumShf) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter Group:			
2	Shift amount	Output	Binary(4)
3	Error code	I/O	Char(*)
Returned Value:			
	Shift amount	Output	Binary(4)

The Retrieve Number of Columns to Shift Scroller (QsnRtvSclNumShf) API returns the default number of columns to shift the scroller by for the Shift Scroller Left (QsnShfSclL) and Shift Scroller Right (QsnShfSclR) APIs. The default number of columns is specified on the session description. See "Create a Session (QsnCrtSsn) API" on page 24-2 and "Change Session (QsnChgSsn) API" on page 24-1 for details.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be queried.

Optional Parameter Group

Shift amount
OUTPUT; BINARY(4)
The variable that contains the number of scroller columns to shift by when the QsnRtvSclNumShf API has completed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Shift amount
OUTPUT; BINARY(4)
Returns the value for the shift amount parameter if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3D6 E Session handle is incorrect.

Retrieve Number of Rows to Roll Scroller (QsnRtvSclNumRoll) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter Group:			
2	Roll amount	Output	Binary(4)
3	Error code	I/O	Char(*)
Returned Value:			
	Roll amount	Output	Binary(4)

The Retrieve Number of Rows to Roll Scroller (QsnRtvSclNumRoll) API returns the default number of rows to roll the scroller by for the Roll Scroller Up (QsnRollSclUp) and Roll Scroller Down (QsnRollSclDown) APIs. The default number of rows is specified on the session description. See "Create a Session (QsnCrtSsn) API" on page 24-2 and "Change Session (QsnChgSsn) API" on page 24-1 for details.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be queried.

Optional Parameter Group

Roll amount
OUTPUT; BINARY(4)
The variable that contains the number of scroller rows to roll by when the QsnRtvSclNumRoll API has completed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Roll amount
OUTPUT; BINARY(4)
This API returns the value for the roll amount parameter if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3D6 E Session handle is incorrect.

Retrieve Session Data (QsnRtvSsnDta) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter Group:			
2	User data pointer	Output	PTR(SPP)
3	Error code	I/O	Char(*)
Returned Value:			
	User data pointer	Output	PTR(SPP)

The Retrieve Session Data (QsnRtvSsnDta) API returns a pointer to the user data for the given session. The user data is the pointer specified on the session description and consists of user-specified data that is associated with the session. See "Format of the Session Description" on page 24-3 for details.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session for which the user data should be returned.

Optional Parameter Group

User data pointer
OUTPUT; PTR(SPP)
A pointer to the user data, as specified on the session description, for the given session.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

User data pointer
OUTPUT; PTR(SPP)
This API returns the value for the user data pointer

parameter if the operation was successful, or the null pointer if an error occurs.

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.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3A4 E Specified window is not active.
CPFA3D6 E Session handle is incorrect.
CPF3C1F E Pointer parameter is not on a 16-byte boundary.

Retrieve Session Description (QsnRtvSsnD) API

Parameters

Required Parameter Group:			
1	Session handle	Input	Binary(4)
2	Session description	Output	Char(*)
3	Length of session description	Input	Binary(4)
Optional Parameter:			
4	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Retrieve Session Description (QsnRtvSsnD) API retrieves a copy of the session description for the given session. The session description may be different from the session description returned when the Create a Session (QsnCrtSsn) or the Change Session (QsnChgSsn) API is called. The following fields will have actual values replacing 0 (if used) :

Number of rows in scroller
Number of columns in scroller
Default number of rows to roll scroller by
Default number of columns to shift scroller by

Required Parameter Group

Session handle
INPUT; BINARY(4)
A handle for the session for which the session description should be returned.

Session description
OUTPUT; CHAR(*)
The session description for the given session. The format of the data is shown in "Format of the Session Description Returned."

Length of session description
INPUT; BINARY(4)
The length of the session description parameter.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Format of the Session Description Returned

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(*)	Session description. The format of the remaining data returned is shown in "Format of the Session Description" on page 24-3.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPFA31E E Required parameter &1 omitted.
CPFA3D6 E Session handle is incorrect.

Roll Scroller Down (QsnRollSclDown) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter Group:			
2	Roll amount	Input	Binary(4)
3	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Roll Scroller Down (QsnRollSclDown) API rolls the scroller down by the specified number of scroller rows. A scroller row is distinct from a scroller line in that a scroller line consists of multiple scroller rows if line wrapping is set on and the line exceeds the width of the scroller.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be rolled.

Optional Parameter Group

Roll amount
INPUT; BINARY(4)
The number of scroller rows to roll the scroller by. If this parameter is omitted or set to 0, the default value is used. The default value can be queried using the Retrieve Number of Rows to Roll Scroller (QsnRtvSclNumRoll) API. If the roll amount would cause the scroller to roll past its top, then the top of the scroller will be displayed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA333 E Parameter &1 not positive integer value.
CPFA31E E Required parameter &1 omitted.
CPFA3D3 E Scroller not printed.
CPFA3D6 E Session handle is incorrect.
CPFA3D8 E Scroller display is not valid.

Roll Scroller Up (QsnRollSclUp) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter Group:			
2	Roll amount	Input	Binary(4)
3	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Roll Scroller Up (QsnRollSclUp) API rolls the scroller up by the specified number of scroller rows. A scroller row is distinct from a scroller line in that a scroller line consists of multiple scroller rows if line wrapping is set on and the line exceeds the width of the scroller.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be rolled.

Optional Parameter Group

Roll amount
INPUT; BINARY(4)
The number of scroller rows to roll the scroller by. If this parameter is omitted or set to 0, the default value is used. The default value can be queried using the Retrieve Number of Rows to Roll Scroller (QsnRtvSclNumRoll) API. If the roll amount causes the scroller to roll past its bottom, then the bottom of the scroller is displayed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The

value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA333 E Parameter &1 not positive integer value.
CPFA31E E Required parameter &1 omitted.
CPFA3D3 E Scroller not printed.
CPFA3D6 E Session handle is incorrect.
CPFA3D8 E Scroller display is not valid.

Shift Scroller Left (QsnShfSclL) API

Parameters

Required Parameter:

1	Session handle	Input	Binary(4)

Optional Parameter Group:

2	Shift amount	Input	Binary(4)
3	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Shift Scroller Left (QsnShfSclL) API shifts the scroller to the left by the specified number of scroller columns. If line wrap mode is on, shifting has no effect.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be shifted.

Optional Parameter Group

Shift amount
INPUT; BINARY(4)
The number of scroller columns to shift the scroller by. If this parameter is omitted or set to 0, the default value is used. The default value can be queried using the Retrieve Number of Columns to Shift Scroller (QsnRtvSclNumShf) API. The scroller is shifted by the minimum of the shift amount and the number of scroller columns between the visible left column and the first column in the scroller.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA333 E Parameter &1 not positive integer value.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.
CPFA3D8 E Scroller display is not valid.

Shift Scroller Right (QsnShfSclR) API

Parameters

Required Parameter:

1	Session handle	Input	Binary(4)

Optional Parameter Group:

2	Shift amount	Input	Binary(4)
3	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Shift Scroller Right (QsnShfSclR) API shifts the scroller to the right by the specified number of scroller columns. Any truncated data will become visible. If line wrap mode is on, shifting has no effect.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be shifted.

Optional Parameter Group

Shift amount
INPUT; BINARY(4)
The number of scroller columns to shift the scroller by. If this parameter is omitted or set to 0, the default value is used. The default value can be queried using the Retrieve Number of Columns to Shift Scroller

(QsnRtvSclNumShf) API. The scroller is shifted by the minimum of the shift amount and the number of scroller columns between the visible right column and the last column (determined by the longest line currently visible) in the scroller.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA333 E Parameter &1 not positive integer value.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.
CPFA3D8 E Scroller display is not valid.

Toggle Line Wrap/Truncate Mode (QsnTglSclWrp) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter Group:			
2	Wrap indication	Output	Char(1)
3	Error code	I/O	Char(*)
Returned Value:			
	Wrap indication	Output	Binary(4)

The Toggle Line Wrap/Truncate Mode (QsnTglSclWrp) API toggles the session between line wrap and truncation mode.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session to be queried.

Optional Parameter Group

Wrap indication
OUTPUT; CHAR(1)
Indicates whether line wrap mode is on or off when the QsnTglSclWrp API has completed. The possible values are:

0 Line wrap mode is now off. Lines are truncated.
1 Line wrap mode is now on.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Wrap indication
OUTPUT; BINARY(4)
This API returns the value for the wrap indication parameter if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.

Chapter 25. Session I/O APIs

The session I/O APIs are:

Backspace on Scroller Line (QsnSclBS) sets the active position to the previous position in the current scroller line.
Go to Next Tab Position in Scroller Line (QsnSclTab) sets the active position to the next horizontal tab position.
Go to Start of Current Scroller Line (QsnSclCR) sets the active position to the start of the current scroller line.
Go to Start of Next Scroller Line (QsnSclNL) sets the active position to the start of the next scroller line.
Print Scroller Data (QsnPrtScl) prints the scroller data.
Read Data from Session (QsnReadSsnDta) reads the data from a session.
Retrieve Session Input Line to Command Line (QsnRtvSsnLin) retrieves the input line from the scroller.
Start New Scroller Line at Current Position (QsnSclLF) sets the active position to the current position on the next scroller line.
Start New Scroller Page (QsnSclFF) starts a new scroller page.
Write Characters to Scroller (QsnWrtSclChr) writes characters to the scroller.
Write Line to Scroller (QsnWrtSclLin) writes a data line to the scroller.

The detailed API descriptions are presented in alphabetical order.

Backspace on Scroller Line (QsnSclBS) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Backspace on Scroller Line (QsnSclBS) API sets the active position to the previous position on the current scroller line. If the active position is at the start of the line, this operation has no effect.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session that the operation applies to.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.

Go to Next Tab Position in Scroller Line (QsnSclTab) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Go to Next Tab Position in Scroller Line (QsnSclTab) API sets the active position to the next horizontal tab position. Each tab interval is eight positions beyond the previous one, starting at the leftmost column in the scroller.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session that the operation applies to.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.

Go to Start of Current Scroller Line (QsnSclCR) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Go to Start of Current Scroller Line (QsnSclCR) API sets the active position to the start of the current scroller line.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session that the operation applies to.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.

Go to Start of Next Scroller Line (QsnSclNL) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Go to Start of Next Scroller Line (QsnSclNL) API sets the active position to the start of the next scroller line.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session that the operation applies to.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.

Print Scroller Data (QsnPrtScl) API

Parameters

Required Parameter:

1	Session handle	Input	Binary(4)

Optional Parameter:

2	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Print Scroller Data (QsnPrtScl) API prints the entire contents of the scroller data to the default printer file. No printer file is produced if the scroller is empty.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session that the operation applies to.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3D3 E Scroller not printed.
CPFA3D6 E Session handle is incorrect.

Read Data from Session (QsnReadSsnDta) API

Parameters

Required Parameter Group:

1	Session handle	Input	Binary(4)
2	Input Buffer	Input	Binary(4)

Optional Parameter Group:

3	Number of bytes read	Output	Binary(4)
4	Error code	I/O	Char(*)

Returned Value:

	Number of bytes read	Output	Binary(4)

The Read Data from Session (QsnReadSsnDta) API is used to read data from a session. A QsnReadInp operation is implicitly performed to read any field data. Data is read from the session input line only. An implicit Clear Field Table (QsnClrFldTbl) operation is issued prior to redefining the session input line on each input operation. The minimum length is the input line length specified on the session description. The window will be rolled up to accommodate this field if required. The data returned consists of only the data entered. That is, only the data from the cursor position within the field up to the last nonblank input character when an AID generating key is pressed. When the QsnReadSsnDta API is processed, an implicit QsnClrFldTbl operation is issued prior to defining the session command line. The only input-capable field defined when the input operation occurs is the command line. The data returned in the input buffer will consist of only the data entered in the session command line.

If an AID key is pressed for which a corresponding function has been defined, this function will be called. Depending upon the return action specified, control would then return to the caller or another input operation will occur. See "Command Key Action Routines" on page 23-1 for details.

Required Parameter Group

Session handle
INPUT; BINARY(4)
A handle for the session from which to read input. The session being read from must be the current window.

You can use the Set Current Window (QsnSetCurWin) API to change the current window.

Input Buffer
INPUT; BINARY(4)
A handle for the input buffer to receive the result of the input operations if a direct operation is specified. The input buffer must have been created with the Create Input Buffer (QsnCrtInpBuf) API. The format of the data returned is the same as that of the Read Input Fields (QsnReadInp) API.

Optional Parameter Group

Number of bytes read
OUTPUT; BINARY(4)
The number of bytes of data read. On a successful read operation, this value is the same as that returned by the Retrieve Length of Field Data in Buffer (QsnRtvFldDtaLen) API if passed the input buffer resulting from this operation.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Number of bytes read
OUTPUT; BINARY(4)
This API returns the value for the number of bytes read parameter if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3A4 E Specified window is not active.
CPFA3D6 E Session handle is incorrect.
CPFA3D9 E Error calling the command key action routine.

Retrieve Session Input Line to Command Line (QsnRtvSsnLin) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Retrieve Session Input Line to Command Line (QsnRtvSsnLin) API retrieves the input line from the scroller that corresponds to the cursor position within the scroller. If the cursor is outside the scroller and the retrieve request directly follows another retrieve with no intervening I/O operations, then the line before the line previously retrieved is returned. Otherwise, the last input line is retrieved. If there is no input data, this API still completes successfully.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session for which to retrieve the input line.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA3A4 E Specified window is not active.
CPFA3D6 E Session handle is incorrect.

Start New Scroller Line at Current Position (QsnSclLF) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Start New Scroller Line at Current Position (QsnSclLF) API sets the active position to the current position on the next scroller line.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session that the operation applies to.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.

Start New Scroller Page (QsnSclFF) API

Parameters

Required Parameter:			
1	Session handle	Input	Binary(4)
Optional Parameter:			
2	Error code	I/O	Char(*)
Returned Value:			
	Return code	Output	Binary(4)

The Start New Scroller Page (QsnSclFF) API starts a new scroller page. Any data currently on the session is rolled off the top of the scroller, but can still be viewed by rolling the scroller up.

Required Parameter

Session handle
INPUT; BINARY(4)
A handle for the session for which to start the new page.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.

Write Characters to Scroller (QsnWrtSclChr) API

Parameters

Required Parameter Group:

1	Session handle	Input	Binary(4)
2	Data	Input	Char(*)
3	Data length	Input	Binary(4)

Optional Parameter:

4	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Write Characters to Scroller (QsnWrtSclChr) API writes one or more characters to the scroller starting at the active position. The active position following this operation is one position past the last character written or that specified by a control character sequence if it appears at the end of the data. If the entire data string cannot fit in the scroller buffer, no portion of the string will be written.

Required Parameter Group

Session handle
INPUT; BINARY(4)
A handle for the session to which the scroller characters are to be written.

Data
Input; CHAR(*)
The characters to be written to the scroller. If the data does not fit within the width of the session window, it is wrapped across multiple lines or truncated, depending on the value of the wrap indication field on the session description.

Data length
Input; CHAR(*)
The length of the data parameter.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code
OUTPUT; BINARY(4)
A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

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.
CPFA333 E Parameter &1 not positive integer value.
CPFA31E E Required parameter &1 omitted.
CPFA340 E Operation not supported with double-byte data.
CPFA3D6 E Session handle is incorrect.
CPFA3D7 E Data for scroller is too long.

Write Line to Scroller (QsnWrtSclLin) API

Parameters

Required Parameter Group:

1	Session handle	Input	Binary(4)
2	Line data	Input	Char(*)
3	Line data length	Input	Binary(4)

Optional Parameter:

4	Error code	I/O	Char(*)

Returned Value:

	Return code	Output	Binary(4)

The Write Line to Scroller (QsnWrtSclLin) API writes a line of data, such as an informational message, to the scroller. The data is written starting at the first position on the next scroller line. The active position after this operation is the start of the next scroller line following the row containing the last data character written, or specified by a control character sequence if one appears at the end of the data. If the entire line cannot fit in the scroller buffer, no portion of the data will be written.

Required Parameter Group

Session handle
INPUT; BINARY(4)
A handle for the session to which the scroller line is to be written.

Line data
Input; CHAR(*)
The data to be written to the scroller. If the line does not fit within the width of the session window, it is wrapped across multiple lines or truncated, depending on the value of the wrap indication field on the session description.

Note: The first 2 bytes of the scroller are reserved for the prefix area to the left of the scroller line.

Line data length
Input; CHAR(*)
The length of the line data parameter.

Note: The first 2 bytes of the scroller are reserved for the prefix area to the left of the scroller line.

Optional Parameter

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structures, see "Error Code Parameter" on page 2-9. If this parameter is omitted, all diagnostic and escape messages are issued to the application.

Returned Value

Return code

OUTPUT; BINARY(4)

A return code indicating the result of the operation. The value returned will be 0 if the operation was successful, or -1 otherwise.

Error Messages

CPD0024 E No matching shift-in character for shift-out character.
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.
CPFA333 E Parameter &1 not positive integer value.
CPFA31E E Required parameter &1 omitted.
CPFA3D6 E Session handle is incorrect.
CPFA3D7 E Data for scroller is too long.
CPG3264 D DBCS character string does not have even length.

Performance Considerations

Specifying EBCDIC control-character options on the session description can incur overhead. Additional processing is required to handle these. Specifying the scroller line and character display as immediate can incur additional overhead. An output operation will occur for each line or group of characters written. If you need to write multiple lines to the scroller, you can achieve better performance by delaying line display until all the lines are written. Then you can use the Display Scroller Bottom (QsnDspSclB) API to display the data on the screen.

Create Session and Read Data Example

The sample program in Figure 25-1 shows how to create and read data from a session. The resulting screen output is shown in Figure 25-2 on page 25-8.

```
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "qsnsess.h"

#define TRUE    1
#define FALSE   0

#define PF1 "F4=Move  F5=Resize  "
#define PF2 "PF6=Print  "

typedef struct {
    Qsn_Ssn_Desc_T sess_desc;
    char           buffer[100];
} storage_t;

int main (void)
{
    Qsn_Inp_Buf_T ibuf = 0;
    int i;
    char text[100];
    storage_t    storage;

    Qsn_Ssn_T       session1;
    Qsn_Ssn_Desc_T  *sess_desc = (Qsn_Ssn_Desc_T *) &storage;
    Qsn_Win_Desc_T  win_desc;
    Q_Bin4          win_desc_length = sizeof(win_desc);
    char           *pf1 = PF1;
    Q_Bin4          pf1_len = sizeof(PF1) - 1;
    char           *pf2 = PF2;
    Q_Bin4          pf2_len = sizeof(PF2) - 1;
    Q_Bin4          sess_desc_length = sizeof(Qsn_Ssn_Desc_T) +
                                       pf1_len + pf2_len;

    QsnInzSsnD( sess_desc, sess_desc_length, NULL);
    QsnInzWinD( &win_desc, win_desc_length, NULL);

    sess_desc->cmd_key_desc_line_1_offset = sizeof(Qsn_Ssn_Desc_T);
    sess_desc->cmd_key_desc_line_1_len = pf1_len;
    memcpy( storage.buffer, pf1, pf1_len );

    sess_desc->cmd_key_desc_line_2_offset = sizeof(Qsn_Ssn_Desc_T) +
                                            pf1_len;
    sess_desc->cmd_key_desc_line_2_len = pf2_len;
    memcpy( storage.buffer + pf1_len, pf2, pf2_len );

    sess_desc->scl_line_dsp  = '1';
    sess_desc->scl_chr_dsp  = '1';
    sess_desc->num_input_line_rows = 2;
    sess_desc->wrap = '0';

    QsnCrtSsn( sess_desc, sess_desc_length, NULL, 0,  '1',
               &win_desc, win_desc_length, NULL, 0,
               &session1, NULL);

    if (ibuf == 0)
        ibuf = QsnCrtInpBuf(100, 50, 0, NULL, NULL);
    while ( TRUE ) {
        QsnReadSsnDta( session1, ibuf, NULL, NULL);
        if (strncmp(QsnRtvFldDta(ibuf, NULL, NULL), "exit", 4) == 0)
            break;
    }
 }
```

Figure 25-1. Program for Creating a Session and Reading Data

Create Session and Read Data Example

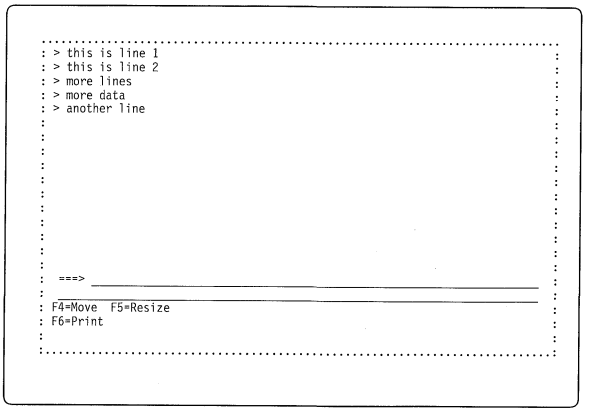


Figure 25-2. Screen Output from Create Session Program

Part 7. Edit Function APIs

Chapter 26. 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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.

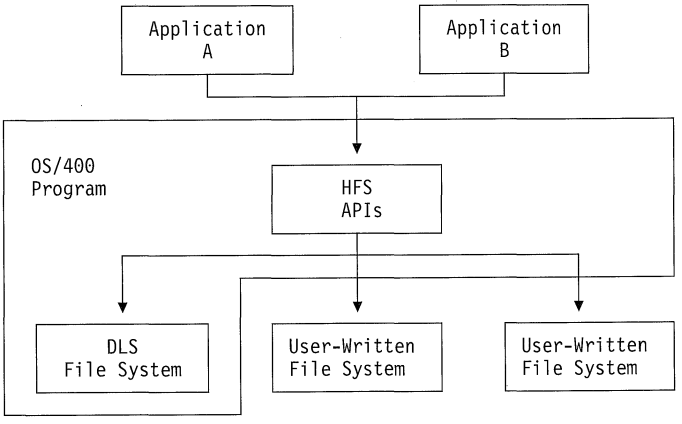
Error Messages

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 8. Hierarchical File System APIs

Chapter 27. 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.

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)

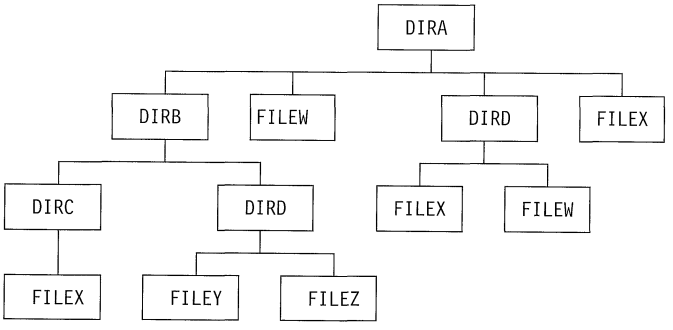
Before using the HFS APIs, read the remaining sections in this chapter. They discuss basic HFS concepts, explain authority to use file systems and their objects, and describe directory entry attributes in detail. Next, if you plan to use the HFS APIs to work with an existing file system, read the documentation provided by the file system for any differences or additional considerations. If you are working with the document library services (DLS) file system, refer to the *Office Services Concepts and Programmer's Guide.* 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 29, "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 29-6. 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 by the file system 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 and 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 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.

- 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 some APIs.

QWRTDTTM
CHAR(13)
The date and time of day the file or directory was last written to, in CYYMMDDHHMMSS format.

| Changes to this attribute may not be supported by all file
| systems. Refer to the file system's documentation for
| any restrictions.

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 28-17. For details about the interface between the QHFRDDR API and a new file system, see page 29-24.

| **Note:** The DLS file system may return this attribute
| when the Retrieve Directory Entry Attributes
| (QHFRTVAT) API is called.

Other Directory Entry Attributes

File systems can define their own unique attributes for files and directories in addition to the standard ones. The file system's documentation defines the attributes' names and values, and explains how to access and use them.

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 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[1]
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[1]

Note:

1 The attribute value is either a character or the character representation of a binary field.

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.

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

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 28. Hierarchical File System APIs

This chapter describes the HFS APIs that allow you to interface 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 27-2. For the format of the table, see "Attribute Information Table" on page 27-4.

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 informa-

[1] Changing directory entry attributes in the DLS file system requires *CHANGE authority to the objects to which the attributes apply.

tion 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.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 28-22 or "Write to Stream File (QHFWRTSF) API" on page 28-23.

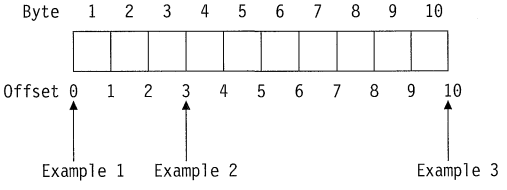
For brief examples of how the move information and the distance to move work together, see "How to Move the File Pointer."

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.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.
CPF1F06 E Directory in use.
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.
| CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

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

| CPF1F06 E Directory 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.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.

[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.

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.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 27-2.

If no attributes are specified, the file system's defaults are used for required information.

---

[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). 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 28-1.

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.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.

[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.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.
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.
CPF1F37 E File is a read-only 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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.

[6] Deleting a DLS file requires *ALL authority to that 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

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.
CPF1F82 E Function not supported.
CPF1F86 E Force all files did not complete successfully.
CPF1F87 E Missing or damaged exit program &2.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Registered File Systems (QHFLSTFS) API

Parameters

Required Parameter Group:

1	Qualified user space and library	Input	Char(20)
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

Qualified 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		Type	Information
Dec	Hex		
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		Type	Information
Dec	Hex		
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 signaled. 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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 other 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.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 28-20.

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.
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.

[7] Moving a DLS file requires *ALL authority to the source file and *CHANGE authority to both the source and the target directories.

| CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 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.

[8] Opening a directory in the DLS file system requires *USE authority to the directory.

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 27-2. For the format of the table, see "Attribute Selection Table" on page 27-4.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 27-3 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.

Do not use the QHFOPNSF API to change the directory entry attributes for an existing file (position 0). Instead, see "Change Directory Entry Attributes (QHFCHGAT) API" on page 28-1.

When the file is opened, the file pointer is set to the first byte of the file (position 0). Subsequent read/write operations move or increase the value of the file pointer. To move it

[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.

explicitly, see "Change File Pointer (QHFCHGFP) API" on page 28-2.

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 28-16.

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 28-16.

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 28-4, 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 27-4.

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 27-2. If you specify other standard attributes, they are ignored.
- The DLS-defined attributes file type, text description, and profile GCID, described in the *Office Services Concepts and Programmer's Guide*. 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 opened. The lock mode you specified when opening a stream file restricts open operations by other jobs only; it does not restrict additional open operations by the job that locked the file.

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

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 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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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	Output	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 28-13.

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

[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.

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 27-2.

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.
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 27-4. 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.
| CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

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 (position 0). You can move the pointer explicitly with the QHFCHGFP API; see "Change File Pointer (QHFCHGFP) API" on page 28-2 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.
| CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

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. (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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.

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.

[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.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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	Char(*)
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 28-17 and "Open Directory (QHFOPNDR) API" on page 28-13.

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 27-2. For the format of the table, see "Attribute Selection Table" on page 27-4.

Note: In the DLS file system:

[13] Retrieving directory entry attributes for DLS file system objects requires *USE authority to the object to which the attributes apply.

- 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.
- 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
  - The error that occurred, given in the QERROR attribute

  For details about these attributes, see "Directory Entry Attributes" on page 27-2.

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 27-4.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 discusses these restrictions.

Required Parameter Group

Open file handle
INPUT; CHAR(16)
The handle returned from the Open Stream File (QHFOPNSF) API when the file was opened. 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 file system may 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.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Write to Stream File (QHFWRTSF) API

Parameters

Required Parameter Group:

1	Open file handle	Input	Char(16)
2	Data buffer	Input	Char(*)
3	Bytes to write	Input	Binary(4)
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 28-2 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.
CPF1F82 E Function not supported.
CPF1F87 E Missing or damaged exit program &2.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

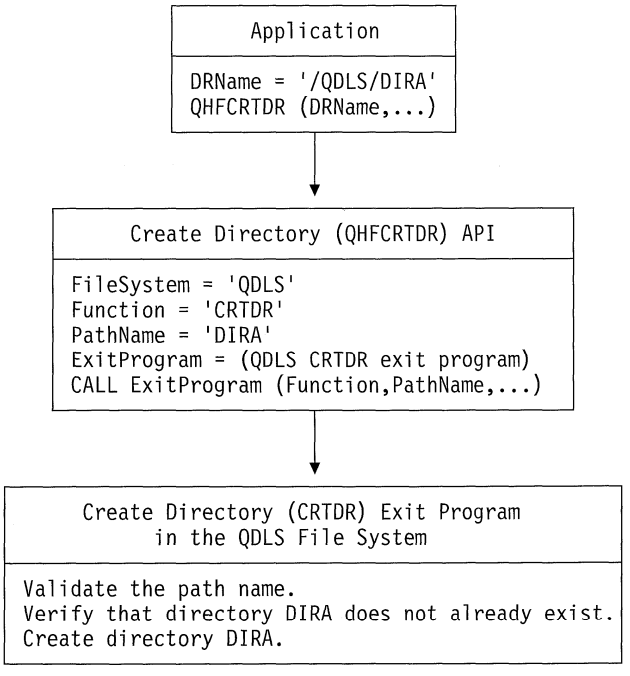
Chapter 29. Preparing to Use New File Systems with the HFS APIs

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 Chapter 28, "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 29-6.

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 29-2 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 29-6 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 29-7 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 29-5 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 29-4 for complete instructions.

5. Give the file system's users authority to the Start Job Session exit program, described on page 29-6. 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 29-6.

- 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 consisting of 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 27-4 and "Attribute Information Table" on page 27-4. 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 qualified exit program names	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 29-5.

Required Parameter Group

File system name

INPUT; CHAR(10)

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 the file system chooses to use. Use the format VxRxMx where x stands for the version, release, and modification levels, respectively. The values indicate the different versions in which HFS enabled new support that the file systems can use. The valid versions are:

V2R1M0 Version 2 Release 1 Modification Level 0
V2R3M0 Version 2 Release 3 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 the source 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 target file system. If the source file system and the target file system cannot perform the operations, the API 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 29-11.

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.

3-6 Reserved. These characters must be blank.

List of qualified exit program names
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. 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 29-6.

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 becomes inaccurate, as it does when the program is recompiled, HFS support updates the pointer. If the system pointer cannot be resolved to the object, an error is returned.

You can change and recompile any exit program without having to reregister your file system. You should remember that if others are using the file system, they will have a *SHRRD lock on the Start Job Session exit program. This may prevent recompilation of that program. You should only recompile the Start Job Session program when no one else is using the file system. You can use the Work with Object Locks (WRKOBJLCK) command on the Start Job Session program to see if any other job is using the file system.

Error Messages

CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F52 E Error code not valid.
CPF1F74 E Not authorized to object.

CPF1F81 E API specific error occurred.
CPF1F85 E Not authorized to file system &1.
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.
CPF1F95 E Required exit program not specified.
CPF1F96 E Version level &2 not valid.
CPF1F97 E File system &1 in use.
CPF1F98 E Registration or deregistration cannot be done now.
CPF1F99 E Register information value not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 29-7 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

CPF1F41 E Severe error occurred while addressing parameter list.
CPF1F52 E Error code not valid.
CPF1F81 E API specific error occurred.
CPF1F85 E Not authorized to file system &1.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 28-1.
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 29-2. 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 29-3 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 28-2.
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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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)

Optional Parameter Group:

4	Close type	Input	Char(1)

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 28-3.
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)

Optional Parameter Group: If your file system was registered with a Version 2 Release 3 Modification Level 0, this parameter is passed to your exit program for the Close Directory API.

Close type
INPUT; CHAR(1)
The type of close operation to be performed. Valid values are:

0 If the exit program cannot close the directory, HFS does not mark the directory as closed.
1 Unconditional close. HFS marks the directory as closed regardless of what valid exception is returned by the exit program. HFS may call your exit program for an unconditional close at the end of the job and during the reclaim resource processing. If your file system is called to unconditionally close a directory, the directory should not be marked as open by the file system when control is returned to HFS.

HFS uses the close type value 0 when the QHFCLODR API is called to close the directory. The unconditional close type value 1 is used when:

- The job ends.
- Reclaim resource processing is done.
- The Deregister File System (QHFDRGFS) API is called, and the job is using the file system that is to be deregistered.

API Functions: In addition to the standard functions described on page 29-2, 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 29-3 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.

| CPF1F06 E Directory in use.
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)

| Optional Parameter Group:

4	Close type	Input	Char(1)

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 28-4.
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 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)

| **Optional Parameter Group:** If your file system was registered with a Version 2 Release 3 Modification Level 0, this parameter is passed to your exit program for the Close Stream File API.

| **Close type**
INPUT; CHAR(1)
The type of close operation to be performed. Valid values are:

| **0** If the exit program cannot close the file, HFS does not mark the file as closed.
1 Unconditional close. HFS marks the directory as closed regardless of what valid exception is returned by the exit program. HFS may call your exit program for an unconditional close at the end of the job and during the reclaim resource processing. If your file system is called to unconditionally close a file, the file should not be marked as open by the file system when control is returned to HFS.

| HFS uses the close type value 0 when the QHFCLOSF API is called to close the file. The unconditional close type value 1 is used when:

- The job ends.
- Reclaim resource processing is done.
- The Deregister File System (QHFDRGFS) API is called, and the job is using the file system to be deregistered.

API Functions: The QHFCLOSF API performs the standard functions described on page 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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.

| CPF1F06 E Directory in use.
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 28-4.
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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3.

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 failure.
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 28-5.
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)

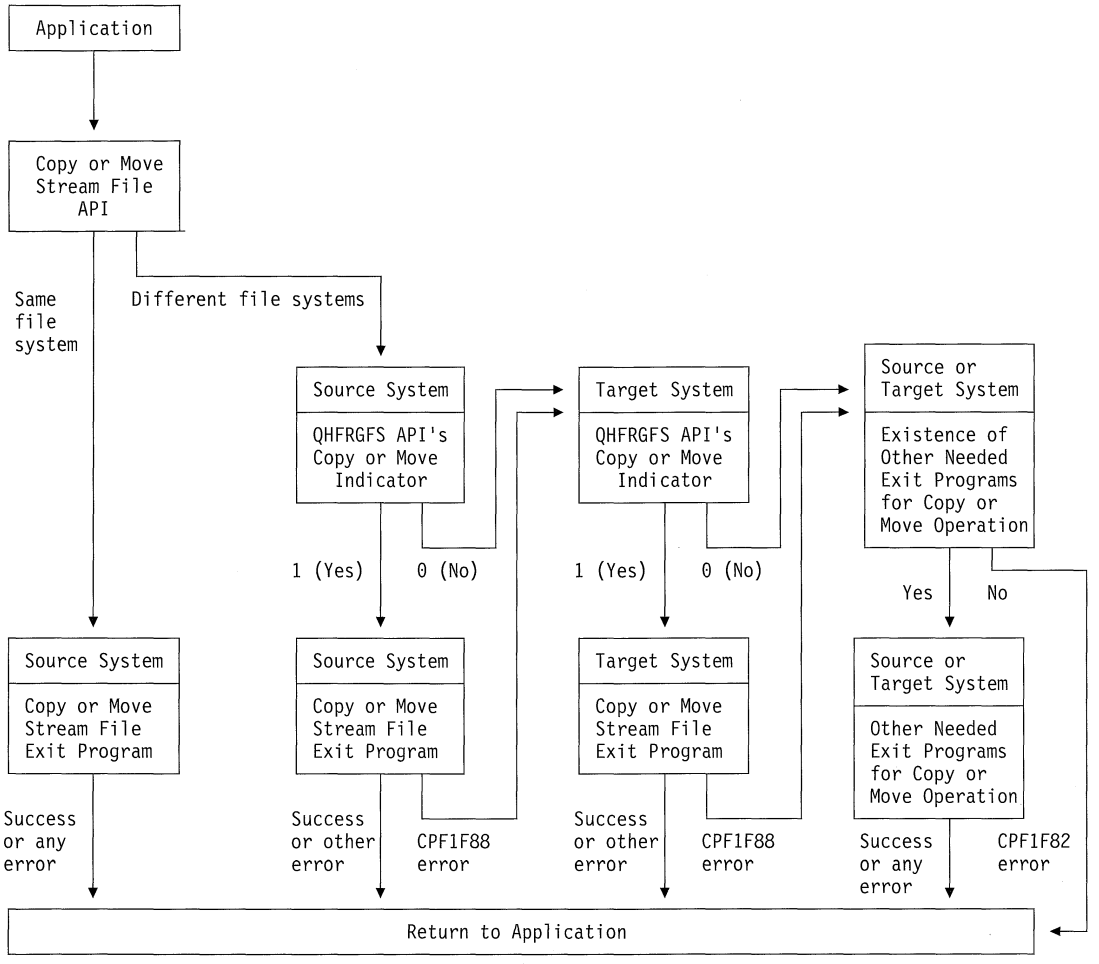
Target file path name
INPUT; CHAR(*)
The API removes the file system name before passing the path name to the exit program.

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 29-2.

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:

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 29-4.

   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 source 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 source 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 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 29-3 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 29-12.

    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 28-6.
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 29-2 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 29-3 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 28-7.
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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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 28-8.
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 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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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.
CPF1F37 E File is a read-only 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 28-8.
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 29-2 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 29-3.

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 28-9.
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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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 28-10.
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 29-2 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 29-3 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 28-12.
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 29-2.

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 29-12, 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 29-3 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 29-14.
- 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 28-13.
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 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 29-2. 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 29-3 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 28-14.
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 29-2 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 29-3 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 28-17.
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 29-2 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 29-3 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 28-19.
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 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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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 28-19.
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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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 28-20.
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 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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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 28-21.
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 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 29-2. 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 29-3 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 28-22.
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 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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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 28-23.
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 29-2.

Exit Program Requirements: You must create an exit program that performs the standard functions described on page 29-3 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 9. High-Level Language APIs

Chapter 30. Application Development Manager/400 APIs

The Application Development Manager/400 APIs allow a control language (CL) command such as the Build Part command (BLDPART) 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.

There are four APIs presented in alphabetical order in this chapter:

- Get Space Status
- Read Build Information
- Set Space Status
- Write Build Information.

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 30-4.

The following compilers and preprocessors use these APIs.

Figure 30-1. 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
ILE C	CRTCMOD, CRTBNDC	N/A
ILE SQL C	CRTSQLCI	N/A
ILE SRVPGM	CRTSRVPGM	N/A
ILE CRTPGM	CRTPGM	N/A
CRTPNLGRP	CRTPNLGRP	N/A
CRTMNU	CRTMNU TYPE(*UIM)	N/A

Note: The CRTMNU command does not write to the API when TYPE is *DSPF or *PGM. Users can create user-defined types to supplement the types supported by the Application Development Manager/400 product. User compilers and preprocessors can now write to this API and be recognized by the BLDPART command.

The following diagram shows the proper usage and order in which the APIs should be called.

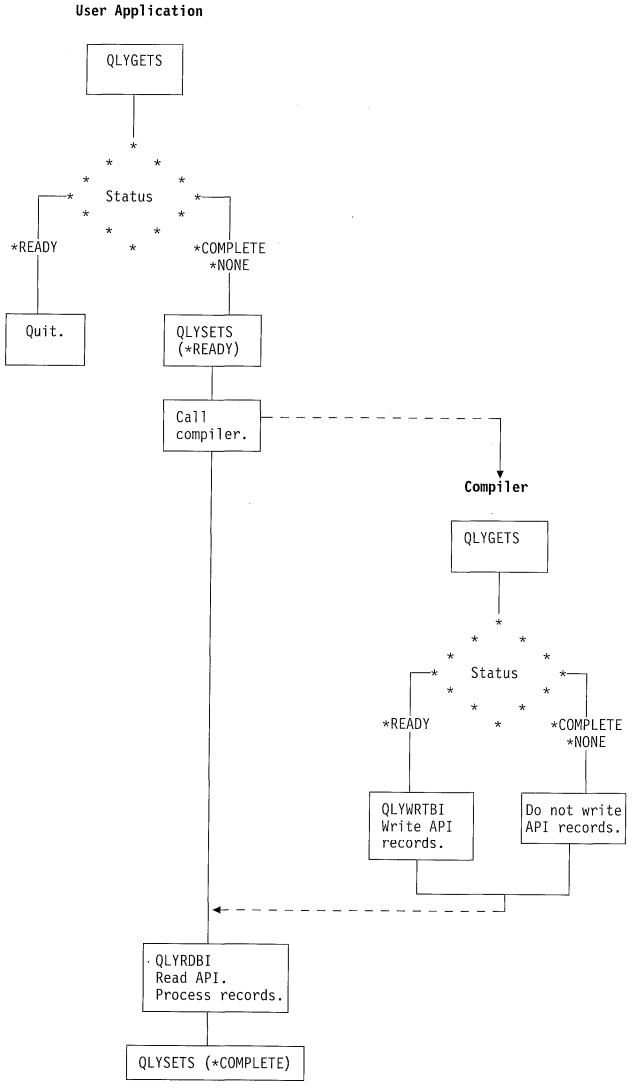


Figure 30-2. Overall Application Development Manager/400 API Usage

QLYGETS should be called by the application or compiler before 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 30-3. 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.

| CPF9872 Program &1 in library &2 ended. Reason code &3.

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 final 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 maximum size parameter is not valid.[1]
LIB9006	Value specified for read mode parameter is not valid.
LIB9007	Value specified for maximum size parameter is too small.
LIB9009	Build information 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.
CPF9872	Program &1 in library &2 ended. Reason code &3.

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 pre-processors, 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

[1] The LIBxxxx error messages are located in the message file QLIBMSG in the QSYS library.

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.
CPF9872	Program &1 in library &2 ended. Reason code &3.

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.

QLYWRTBI continues to write records to the API space concatenated to previous records written, until QLYSETS is called. See "Record Types" for the records that can be written. See "Examples of Records Written" on page 30-16 for examples of the sequence of records 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.

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	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	Build information space does not exist, or it is damaged or deleted.
CPF3CF1	Error code parameter not valid.
CPF9872	Program &1 in library &2 ended. Reason code &3.

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 member start record
- Processor object start record
- Normal processor end record
- Normal processor end call next record
- Normal multiple end record
- Abnormal processor end record
- Include record

File reference record
Module reference record
Service program reference record
Bind directory 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.

All fields where information is not available to put in these records should be filled with blanks.

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 30-4. Record Types and Processors (Part 1)

Record Type	RPG/400: CRTRPGPGM	COBOL/400: CRTCBLPGM	C/400: CRTCPGM	CLD: CRTCLD	DDS: CRTPF CRTLF CRTDSPF CRTICFF CRTPRTF	CL(5): CRTCLPGM	CMD: CRTCMD	SQL/400: CRTSQLRPG CRTSQLCBL CRTSQLC
Processor member start	X(1 3)	X	X(3)	X(1 3)	X	X(1 3)	X(1)	X(1)
Processor object start								
Normal processor end	X	X	X	X	X	X	X	X
Normal processor end call next								X
Normal multiple end record								
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)
Module reference								
Service program reference								
Bind directory reference								
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(10)	X	X(10 13)		X(10)	X(1 4)		X(1)
Object already exists error					X			
Start of new program		X						

Figure 30-5 (Page 1 of 2). Record Types and Processors (Part 2)

Record Type	ILE C: CRTCMOD CRTBNDC	ILE SQL C: CRTSQLCI	ILE SRVPGM: CRTSRVPGM	ILE CRTPGM	UIM: CRTPNLGRP	CRTMNU	UDT(19): SYSTYPE(*NONE)	UDT: member
Processor member start	X(3)	X(1)	X(18)		X	X(17)		X
Processor object start				X(16)			X	
Normal processor end	X	X	X	X	X	X		X
Normal processor end call next	X(14)	X						X
Normal multiple end record							X	
Abnormal processor end	X	X	X	X	X	X	X	X
Include	X(8 12)	X(1 7)			X	X		X
File reference	X	X(1)						X
Module reference			X	X				
Service program reference			X	X				

Figure 30-5 (Page 2 of 2). Record Types and Processors (Part 2)

Record Type	ILE C: CRTCMOD CRTBNDC	ILE SQL C: CRTSQLCI	ILE SRVPGM: CRTSRVPGM	ILE CRTPGM	UIM: CRTPNLGRP	CRTMNU	UDT(19): SYSTYPE(*NONE)	UDT: member
Bind directory reference		X	X	X				
Record format reference	X	X(1)						X
Field reference								X
Message reference					X	X		X
External reference error	X(10 13)	X(1)	X(15)	X(15)	X	X		X
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 member 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 referenced 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 compilers do 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 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 is set to *LIBL because no specific library can be determined if the file is not found in the library list.
11. RPG/400 compiler 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 contains 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 is set to *CURLIB because no specific library can be determined if the file is not found in the library list.
14. This record is written only by the CRTBNDxxx commands.
15. This record is written only when a SRVPGM or MODULE does not exist, and this causes the compilation to fail.
16. The object fields in this record refer to the ENTMOD parameter for CRTPGM.
17. CRTMNU only writes records when TYPE=*UIM.
18. The source used fields contain the same information as the source specified fields.
19. User-defined types are part types that were created in addition to the part types available with the Application Development Manager/400 product. See *SAA* AD/Cycle* Application Development Manager/400 User's Guide*, SC09-1376 for more information on creating and using user-defined part types.

Processor Member Start Record

This, or the Processor object start record, must be the first record that is passed by the compiler or preprocessor on its first call to the QLYWRTBI API. Its purpose is to identify the source that is being compiled, and also to describe the expected output object, if any.

Note: This record was previously called the "processor start" record, but the format remains the same.

The Processor member start 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)	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 applies: All compilers and preprocessors listed in Figure 30-1 on page 30-1 except CRTPGM, and user-defined types added with SYSTYPE(*NONE) on the ADDADMTYPE command.

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 specified. The library name of the source file specified on the compiler or preprocessor command.

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 member name specified. The source member name 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 object name specified. The object 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 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" on page 30-8.)

Target object type. The AS/400 type of the object to be created (for example, *FILE).

Processor Object Start Record

This, or the Processor member start record, must be the first record that is passed by the compiler or preprocessor on its first call to the QLYWRTBI API. Its purpose is to identify the object that is being processed, and also to describe the expected output object, or, for user-defined types, the expected location of the output members, if any.

User-defined types added with SYSTYPE(*NONE) on the ADDADMTYPE command must write this record before any other record.

The Processor object start 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)	Processor command
18	12	CHAR(10)	Object name specified
28	1C	CHAR(10)	Object library name specified
38	26	CHAR(7)	Object type specified
45	2D	CHAR(10)	Object name used

Offset			
Dec	Hex	Type	Field
55	37	CHAR(10)	Object library name used
65	41	CHAR(7)	Object type used
72	48	CHAR(10)	Target object name specified
82	52	CHAR(10)	Target object library name specified
92	5C	CHAR(7)	Target object type specified
99	63	CHAR(1)	Reserved

Processors for which this record type applies: CRTPGM and user-defined types added with SYSTYPE(*NONE) on the ADDADMTYPE command.

Field Descriptions

Object library name specified. The library name of the object specified on the compiler or preprocessor command. If the object type specified is a user-defined type with SYSTYPE(*NONE), the library name specified should be the group library name.

Object library name used. The actual name of the library that the object was found in. The library name could be different from the specified library name because, for example, *LIBL or *CURLIB was specified. This field contains the name the library resolves to.

Object name specified. The object name specified on the command. If the object type specified is a user-defined type with SYSTYPE(*NONE), the object name specified should be the part name.

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.

Object type specified. The object type specified on the command. For user-defined types this must be left blank. If the object type specified is a user-defined type with SYSTYPE(*NONE), the object type specified should be the part type.

Object type used. The actual type of the object used. For example, *MODULE. For user-defined types this can be left blank.

Processor command. The compiler or preprocessor that wrote this record, for example, CRTPGM.

Record length. The length of this record is 100.

Record type. The type of this record is '50'.

Reserved. An ignored field.

Target object library name specified. The library of the target object specified on the command. For user-defined types, the library where the output members are created, as specified on the command.

Target object name specified. The name of the object to be created, or modified as specified on the command. For user-defined types this can be left blank.

Target object type specified. The type of the object to be created. For example, *PGM. The actual name of the object that was created is passed through the Normal processor end record. (See "Normal Processor End Record.") For user-defined types, the names of the output members are passed through the Normal multiple end record. For user-defined types this value must be *MBR.

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			
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)	Object name created
18	12	CHAR(10)	Library
28	1C	CHAR(7)	Object type
35	23	CHAR(10)	Member
45	2D	CHAR(7)	Message identifier

Processors for which this record type applies: All compilers and preprocessors listed in Figure 30-1 on page 30-1, except user-defined types added with SYSTYPE(*NONE) on the ADDADMTYPE 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 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 CRTSQLCI command is entered with OPTION(*GEN), and the member is created successfully, the last record written by CRTSQLCI is the Normal processor end call next record. The preprocessor then calls CRTCPGM that eventually writes the normal or abnormal processor end record.

The Normal processor end call next 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)	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 applies:
CRTSQLC and CRTSQLCI if OPTION(*GEN) is specified on the command, CRTBNDC.

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.

Normal Multiple End Record

This is the last record passed by a user-defined type added with SYSTYPE(*NONE) on the ADDADMTYPE command. It identifies normal multiple end processing of **all** the output members. One normal multiple end record is written per member generated. The normal processor end record should not be written.

Note: It is possible that the processor generated 10 members on the last build, and because of a change, now needs to regenerate just 2 of those members. For the build process to preserve the relationships to the remaining 8 members, the processor must write all members to the API, regardless of whether the member was actually regenerated. The build process ignores those parts (members) that have either not changed (because the processor did not regenerate them), or do not exist (because the processor did not generate them, and they may exist higher in the hierarchy).

The Normal multiple 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(10)	Library
18	12	CHAR(10)	File name created
28	1C	CHAR(10)	Member
38	26	CHAR(32)	Part type
70	46	CHAR(32)	Part language
102	66	CHAR(22)	Reserved

Processors for which this record type applies: User-defined types added with SYSTYPE(*NONE) on the ADDADMTYPE command.

Field Descriptions

File name created. The file name that was created or used to hold the member.

Library. The library where the member was created.

Member. The name of the member created

Part language. The language of the part to represent this member.

Part type. The type of the part to represent this member.

Record length. The length of this record is 124.

Record type. The type of this record is '65'.

Reserved. An ignored field.

Abnormal Processor End Record

This is the last record passed if the compiler or preprocessor fails because of an error. For example, an object or member was not created because of 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, message file, module, bind directory or service program could not be found, the command passes the External reference error record before passing this one. See "External Reference Error Record" on page 30-14 for more information on this record.

The Abnormal 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(7)	Message identifier
15	F	CHAR(1)	Reserved

Processors for which this record type applies: All compilers and preprocessors listed in Figure 30-1 on page 30-1.

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		Type	Field
Dec	Hex		
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 applies:
CRTSQLC, CRTRPGPGM, CRTCBLPGM, CRTCPGM, CRTRPGMOD, CRTCMOD, CRTBNDC, CRTPNLGRP, CRTMNU, CRTSQLCI, and user-defined types represented as members.

Field Descriptions

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 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 from the one specified in the source as a result of an override. This name must always be filled in.

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 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 from the one specified in the source, as a result of an override).

Include file member specified. The name of the source member containing the include, as specified in the source.

Include file member used. The actual name of the source member containing the include that was used. This name must always be filled in.

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 applies:
CRTSQLC, CRTRPGPGM, CRTCBLPGM, CRTCPGM, CRTCMOD, CRTBNDC, CRTPF, CRTLF, CRTPRTF, CRTDSPF, CRTICFF, CRTCLPGM, CRTSQLCI, and user-defined types represented as members.

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 specified. The name of the file referenced, as specified in the source.

File name used. The name of the actual file that was referenced. This name must always be filled in.

File library name specified. The name of the library 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.

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.

Module Reference Record

This record is passed when a module is successfully referenced by a processor. This record is not to be written for the ENTMOD module, on the CRTPGM command.

The Module 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)	Module name specified
18	12	CHAR(10)	Module library name specified
28	1C	CHAR(10)	Module name used
38	26	CHAR(10)	Module library name used

Processors for which this record type applies:
CRTSRVPGM and CRTPGM.

Field Descriptions

Module name specified. The name of the module referenced, as specified on the command, or in the bind directory.

Module name used. The name of the actual module that was referenced. This name must always be filled in.

Module library name specified. The name of the library of the module referenced, as specified on the command, or in the bind directory.

Module library name used. The name of the actual library that contains the module that was referenced. The library name could be different from the specified library name because *LIBL or *CURLIB was specified.

Record length. The length of this record is 92.

Record type. The type of this record is '55'.

Reserved. An ignored field.

Service Program Reference Record

This record is passed when a service program is successfully referenced by a processor.

The Service program 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)	Service program name specified
18	12	CHAR(10)	Service program library name specified
28	1C	CHAR(10)	Service program name used
38	26	CHAR(10)	Service program library name used
48	30	CHAR(16)	Service program signature used

Processors for which this record type applies:
CRTSRVPGM and CRTPGM.

Field Descriptions

Record length. The length of this record is 64.

Record type. The type of this record is '60'.

Service program name specified. The name of the service program as specified on the command.

Service program name used. The name of the actual service program that was referenced. This name must always be filled in.

Service program library name specified. The name of the library of the service program referenced, as specified on the command.

Service program library name used. The name of the actual library that contains the service program that was referenced. The library name could be different from the specified library name because *LIBL or *CURLIB was specified.

Service program signature used. The current signature of the service program used.

Bind Directory Reference Record

This record is passed when a module is successfully referenced by a processor. This record is not to be written for the ENTMOD module, on the CRTPGM command.

The Bind directory 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)	Bind directory name specified
18	12	CHAR(10)	Bind directory library name specified
28	1C	CHAR(10)	Bind directory name used
38	26	CHAR(10)	Bind directory library name used

Processors for which this record type applies:
CRTSRVPGM and CRTPGM.

Field Descriptions

Bind directory name specified. The name of the bind directory referenced, as specified on the command.

Bind directory name used. The name of the actual bind directory that was referenced. This name must always be filled in.

Bind directory library name specified. The name of the library of the bind directory referenced, as specified on the command.

Bind directory library name used. The name of the actual library that contains the bind directory that was referenced. The library name could be different from the specified library name because *LIBL or *CURLIB was specified.

Record length. The length of this record is 48.

Record type. The type of this record is '75'.

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, SQL/400, ILE RPG, ILE C or ILE SQL C processors whenever a declara-

tion 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		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)	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: CRTSQLC, CRTPF, CRTLF, CRTDSPF, CRTICFF, CRTPRTF, CRTCLPGM, CRTRPGPGM, CRTCBLPGM, CRTCPGM, CRTCMOD, CRTBNDC, CRTSQLCI, and user-defined types represented as members.

Field Descriptions

File name specified. The name of the file being referenced, as specified in the source.

File name used. The name of the actual file that was referenced. This name must always be filled in.

File library name specified. The name of the library 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.

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		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)	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 applies: CRTPF, CRTLF, CRTPRTF, CRTDSPF, and CRTICFF.

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 specified. The name of the file being referenced, as specified in the source.

File name used. The name of the actual file that was referenced. This name must always be filled in.

Fixed/variable length indicator. Contains F if the field is of fixed length, V if variable length.

File library name specified. The name of the library 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.

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
56	38	BINARY(4)	Nesting Level

Processors for which this record type applies: CRTPF, CRTLF, CRTDSPF, CRTPRTF, CRTICFF, CRTCMD, CRTPNLGRP, CRTMNU, and user-defined types represented as members.

Field Descriptions

Message file library specified. The name of the library that contains the message file, as specified in the source.

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 name specified. The name of the message file referenced, 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 identifier. The message ID referenced.

Nesting Level. The level of nesting of the MSGF. MSGFs referenced in the root source have a nesting level of 1, MSGFs found in level 1 have a nesting level of 2 and so on.

Record length. The length of this record is 60.

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, message file, module, bind directory or service program 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 30-10).

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 applies:
CRTSQLC, CRTPF, CRTLF, CRTDSPF, CRTICFF, CRTPRTF, CRTCLPGM, CRTRPGPGM, CRTCBLPGM,

CRTCPGM, CRTCMOD, CRTBNDC, CRTSQLCI, CRTSRVPGM, CRTPGM, CRTPNLGRP, CRTMNU, and user-defined types represented as members.

Field Descriptions

Object library name specified. The name of the library that contains the object that was not found.

Object library name used. The actual name of the library that contains the object that was referenced.

Object name specified. The name of the object referenced that was not found.

Object name used. The actual name of the object that was referenced. This name must always be filled in.

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 30-10).

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 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 applies: CRTPF and CRTLF.

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 is able 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)	Object name created
28	1C	CHAR(10)	Object library name
38	26	CHAR(7)	Message identifier
45	2D	CHAR(3)	Reserved
48	30	CHAR(7)	Object type
55	37	CHAR(1)	Reserved

Processors for which this record type applies: CRTCBLPGM.

Field Descriptions

Message identifier. The message ID of the completion message.

New program name. The name of the new program, per IDENTIFICATION DIVISION.

Object library name. The library where the object was created. This field contains blank if an error occurred.

Object name created. The name of the object created in the previous step. If an object was not created because of

syntax errors or because REPLACE(*NO) was specified and the object already existed, this field contains '*ERROR'.

Object type. The type of object created. For example, *PGM or *MODULE.

Record length. The length of this record is 56.

Record type. The type of this record is '40'.

Reserved. An ignored field.

Examples of Records Written

The following examples illustrate how compilers and pre-processors communicate with the Application Development Manager/400 APIs in different circumstances. In all these examples, assume 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 pre-processor.

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 member 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 member 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 member 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 member start**
2. **External reference error** (The name of the *Library specified* passed to QLYWRTBI is *LIBL.)
3. **Abnormal processor end**

Example 5

CRTPGM binder successfully binds objects from 2 modules.

The compiler 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 object start**
2. **Module reference**
3. **Module reference**
4. **Bind directory reference**
5. **Service program reference**
6. **Normal processor end**

Chapter 31. 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-33 and "Error Handler for Example COBOL Program" on page A-34 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.

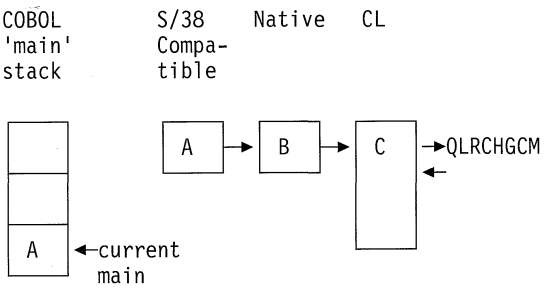
COBOL Program B is a menu program that calls CL Program C.

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.

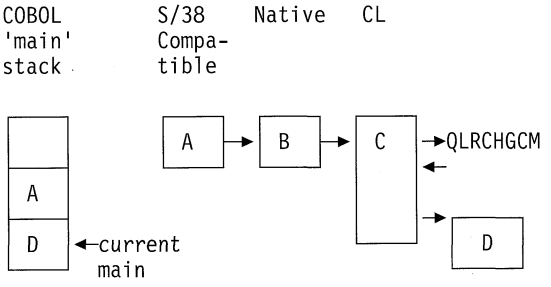
When program C calls the new COBOL application in the form of Program D, Program D becomes the main program in a new run unit. When Program D's run unit ends, control returns to the original run unit, and Program A becomes the current main program again.

If, at the time a run unit is created, a program is active as a subprogram in an existing run unit, and this program is then called within the new run unit, it will be made available in its last-used state.

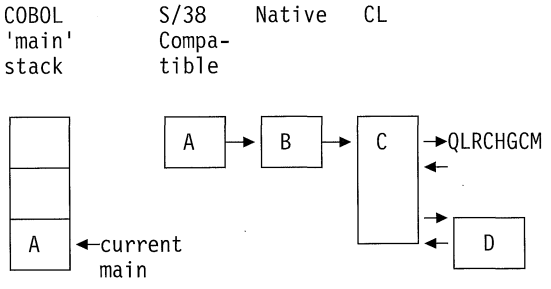
Stage 1



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 10. ILE CEE APIs

Chapter 32. Descriptions of the ILE CEE APIs

Bindable application programming interfaces (APIs) are available with the Integrated Language Environment (ILE) architecture on the OS/400 operating system. In some cases they provide additional function beyond that provided by a specific high-level language. For example, not all high-level languages (HLL) offer intrinsic means to manipulate dynamic storage. In these cases, you can supplement an HLL function by using appropriate bindable APIs. If your HLL provides the same function as a particular bindable API, use the HLL-specific one.

The bindable APIs are useful for mixed-language applications because they are HLL independent. For example, if you use only condition management bindable APIs with a mixed-language application, you will have uniform condition handling semantics for that application. This uniformity can make condition management easier than when using multiple HLL-specific condition handling models.

The bindable APIs provide a wide-range of functions including:

- Activation group and control flow management
- Storage management
- Condition management
- Message services
- Source debugger
- Date and time manipulation
- Math functions
- Call management
- Operational descriptor access

Bindable API Calling Conventions

You access ILE bindable APIs using the same call mechanisms currently used by HLLs to support calls in general.

If you are an ILE C/400 user, you can use an ILE bindable API with the syntax shown in the following example.

```
#include <leawi.h>
main ()
{
 CEExxxx(&parm1, &parm2, ... &parmn, &fc);
}
```

where:

leawi.h
The name of the header file for C prototypes

CEExxxx
The name of the bindable API

parm1, parm2, ... parm*n*
Optional or required parameters passed to or returned from the called bindable API. The & character in the syntax indicates that the parameters are explicitly passed by reference.

fc An optional feedback code that indicates the result of the bindable API. The & character in the syntax indicates that the parameter is explicitly passed by reference.

Naming Conventions of the Bindable APIs

Most bindable APIs are available to any HLL that ILE supports. Naming conventions of the bindable APIs are as follows:

- Bindable API names starting with CEE are intended to be consistent across the IBM SAA systems.
- Bindable API names starting with CEE4 are specific to the AS/400 system.

Data Type Definitions of ILE Bindable APIs

The data type definitions that are used in the syntax of the ILE bindable APIs are described in Figure 32-1.

Figure 32-1 (Page 1 of 2). Data Type Definitions for ILE Bindable APIs

Data Type	Description
_CHAR	A 1-byte unsigned character
_UCHAR	A 1-byte unsigned character
_SCHAR	A 1-byte signed character
_INT2	A 2-byte signed integer
_UINT2	A 2-byte unsigned integer
_INT4	A 4-byte signed integer
_UINT4	A 4-byte unsigned integer
_FLOAT4	A 4-byte single-precision floating-point number
_FLOAT8	An 8-byte double-precision floating-point number
_COMPLEX8	An 8-byte complex number whose real and imaginary parts are each 4-byte single-precision floating-point numbers.
_COMPLEX16	A 16-byte complex number whose real and imaginary parts are each 8-byte double-precision floating-point numbers.
_BITS	A set of adjacent bits within a single storage unit. The notation is `_BITS: x`, where x is the field width in bits. (_BITS may also be used to define unsigned integers.)
_POINTER	A generic pointer
_CHARn	A string (character array) of length n

Figure 32-1 (Page 2 of 2). Data Type Definitions for ILE Bindable APIs

Data Type	Description
_VFLOAT	An ILE variable-length floating-point number used for polymorphic parameter declarations. The length may be any one of 4, 8, or 16-bytes corresponding to single, double, and extended precision.
_VSTRING	An ILE string of arbitrary length used for polymorphic string parameter declarations. The string may be any of a fixed-length string, a null-terminated varying string (known as an "ASCIIZ" string), or a length-prefixed string.
_FEEDBACK	A mapping of the feedback (condition) token
_CEE_ENTRY	A generic entry constant
_HDLR_ENTRY	An entry constant used on the CEEHDLR and CEEHDLU APIs.
_RTX_ENTRY	An entry constant used on the CEERTX and CEEUTX APIs.
_RAGE_ENTRY	An entry constant used on the CEERAGE API.
_CEELABEL	A target label to a code point within a call stack entry.

Strong alignment is assumed in all data structures. Each item is aligned on the proper boundary for its type, with padding if necessary.

Chapter 33. Activation Group and Control Flow APIs

Bindable APIs are provided to manage activation groups and to determine the control flow of procedures.

- Abnormal End (CEE4ABN)
- Find a Control Boundary (CEE4FCB)
- Register Call Stack Entry Termination User Exit Procedure (CEERTX)
- Signal the Termination-Imminent Condition (CEETREC)
- Unregister Call Stack Entry Termination User Exit Procedure (CEEUTX)

The APIs in this chapter are presented in alphabetical order.

Abnormal End (CEE4ABN) API

Syntax

```
void CEE4ABN ([raise_TI], [cel_rc_mod],[user_rc])

             _INT4       *raise_TI;
             _INT4       *cel_rc_mod;
             _INT4       *user_rc;
```

The Abnormal End (CEE4ABN) API abnormally ends the activation group containing the nearest control boundary. The termination-imminent condition can be sent first to give all intervening call stack entries a chance to clean up or stop the abnormal end. This is optional. If the resume cursor is not moved and a resume is requested, then all call stack entries to the nearest control boundary are ended. If the call stack entry for the control boundary is also the oldest call stack entry in the activation group, the activation group ends. The exception message CEE9901 (application error) is sent to the caller of the control boundary, whether or not the activation group ended.

Parameters

raise_TI (input/optional)
Whether or not the terminate-imminent condition should be raised before the end operation.

- **0** The terminate-imminent condition is not raised; the end operation starts immediately. This value is the default if *raise_TI* is omitted.
- **1** The terminate-imminent condition is raised. The end operation occurs only if the handling of the terminate-imminent condition results in the resume cursor not being moved after a resume operation was requested.

cel_rc_mod (input/optional)
A language-specific return code passed from one ILE language to another ILE language. The value and meaning is language-specific.

user_rc (input/optional)
A number representing the user portion of the activation group return code. If this parameter is not supplied, the CEE4ABN API uses the current contents of the activation group return code. If it is supplied, it takes precedence over previously set values.

Conditions

CEE9902	Unexpected user error occurred in &1
Severity: 30	

Usage Notes

- High-level language statements that implement abnormal ending of the activation group do so by calling the CEE4ABN API.
- The job-level return codes are updated whether or not the activation group ended.
- This API cannot end the default activation group.

Find a Control Boundary (CEE4FCB) API

Syntax

```
void CEE4FCB ([ctlbdy_inv], [ctlbdy_type], [fc])

             _INT4       *ctlbdy_inv;
             _INT4       *ctlbdy_type;
             _FEEDBACK   *fc;
```

The Find a Control Boundary (CEE4FCB) API searches the call stack for the nearest call stack entry that is a control boundary. Beginning with the caller of the CEE4FCB API, a search for a control boundary starts from the call stack entry of the caller and progresses to older call stack entries.

Parameters

ctlbdy_inv (output/optional)
A positive number indicating the control boundary call relative to the caller of the CEE4FCB API.

ctlbdy_type (output/optional)
The type of the control boundary found.

- **0** The control boundary found is the oldest call stack entry in the activation group.
- **1** The control boundary found is not the oldest call stack entry in the activation group.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE9902 Severity: 30	Unexpected user error occurred in &1

Register Activation Group Exit Procedure (CEE4RAGE) API

Syntax

```
void CEE4RAGE (procedure, [fc])

            _RAGE_ENTRY *procedure;
            _FEEDBACK   *fc;
```

The Register Activation Group Procedure (CEE4RAGE) API is used to register procedures that are called when an activation group ends. Activation group exit procedures, registered by CEE4RAGE, are called after HLL user exit procedures, but before any system level activation group resource clean up takes place. The procedures are called in the reverse order of their registration.

There is no practical limit to the number of procedures that can be registered. If the same procedure is registered multiple times, it is called multiple times.

Note: This API cannot be called from code running in the default activation group.

Parameters

procedure (input)
An entry variable or constant for the procedure that is to be called at activation group termination.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0257 Severity: 30	The procedure provided for &1 is not valid
CEE3101 Severity: 30	&1 cannot be called in the default activation group
CEE3103 Severity: 30	Cannot allocate storage in &1
CEE3111 Severity: 30	&1 cannot be called at this time
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- The message CEE0257 occurs if *procedure* is not a procedure pointer, or if the procedure identified by *procedure* is not in either the current activation group or the default activation group.
- Once the activation group exit procedures start to run, the CEE4RAGE API cannot be called.

Interface to the Activation Group Exit Procedure

An activation group exit procedure is called when the activation group is ended. Following is the interface to the activation group exit procedure.

Syntax

```
void activation_group_exit(UINT4 *ag_mark,
                           UINT4 *reason,
                           volatile UINT4 *result_code,
                           UINT4 *user_rc)
```

Parameters

ag_mark (input)
The activation group mark that uniquely identifies the activation group within the job.

reason (input)
The reason for the activation group being ended. See Figure 33-1 on page 33-3 for a description of the reason codes.

result_code (input/output)
The value passed as input is the action to be taken as specified by a previous exit procedure. The value passed to the first exit procedure is 0. The output value can specify an action to be taken. If the result code does not match any of the following actions, the output value is ignored and the previous action remains unchanged.

No action

0 Do not change the action.

Recover

10 Do not perform any pending error requests. This is used if a previous exit procedure specified a result code of 20 and a subsequent procedure recovers from the error. The message CEE9901, indicating an application error, is not sent.

Failure

20 Send message CEE9901 to the caller of the control boundary after the remaining exit procedures are called.

21 Send message CEE9901 to the caller of the control boundary. The remaining exit procedures registered by the CEE4RAGE API are not called. This is used if an unrecoverable error occurs in the exit procedure requesting this action.

Note: The application error message CEE9901 is sent after the activation group resources of the system are taken down and the activation group has ended.

user_rc (input/output)
The value passed as input is the *user_rc* returned as output from the previous exit procedure. The value passed to the first exit procedure is 0.

Figure 33-1. Common Reason Codes for Ending Activation Groups and Call Stack Entries.

Bit	Description
Bits 0	Reserved
Bits 1	Call stack entry is canceled because an exception message was sent.
Bits 2-15	Reserved
Bit 16	0 - normal end 1 - abnormal end
Bit 17	Activation Group is ending.
Bit 18	Initiated by the Reclaim Activation Group (RCLACTGRP) command.
Bit 19	Initiated as a result of the job ending.
Bit 20	Initiated by an exit verb, for example exit() in C, or the CEETREC API.
Bit 21	Initiated by an unhandled function check.
Bit 22	Call stack entry canceled because of an out-of-scope jump, for example *longjmp()* in C.
Bits 23-31	Reserved (0)

Register Call Stack Entry Termination User Exit Procedure (CEERTX) API

Syntax

```
void CEERTX (procedure, [token], [fc])

            _RTX_ENTRY  *procedure;
            _POINTER    *token;
            _FEEDBACK   *fc;
```

The Register Call Stack Entry Termination User Exit Procedure (CEERTX) API registers a user-defined procedure that runs when the call stack entry for which it is registered is ended by anything other than a return to the caller.

Parameters

procedure (input)
An entry variable or constant for the procedure that is called if the call stack entry is abnormally ended.

token (input/optional)
A pointer that is passed to *procedure.* If *token* is omitted, a null pointer is passed to *procedure* when *procedure* is called.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0256 Severity: 10	Procedure already registered, registered again
CEE0257 Severity: 30	The procedure provided for &1 is not valid
CEE3103 Severity: 30	Cannot allocate storage in &1
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- Multiple user call stack entry termination procedures can be registered for each call stack entry.
- User call stack entry termination exit procedures run in first in/first out (LIFO) order. If a procedure causes any of the following, the remaining procedures are not run:
  - The call to a call stack entry termination exit procedure fails.
  - An exception is not handled.
  - Control has moved past the invocation of the call stack entry termination procedure. For example, the resume cursor has been moved, or an out-of-scope jump, such as *longjmp()* in ILE C/400, has been used.
- The message CEE0257 occurs only if the pointer contained in the procedure parameter is not a procedure pointer. This does not verify that a call through that pointer will be successful. For example, the call may fail because the activation group containing the procedure no longer exists.

Interface to the Call Stack Entry Termination User Exit Procedure

The following is the interface to the call stack entry termination user exit procedure registered by the CEERTX API.

Syntax

```
void termination_procedure(Term_Token)

            _POINTER    *Term_Token;
```

Parameter

Term_Token (input)
The user-supplied pointer passed on the call to CEERTX that registered the call stack entry termination user exit procedure.

Signal the Termination-Imminent Condition (CEETREC) API

Syntax

```
void CEETREC ([cel_rc_mod], [user_rc])

            _INT4       *cel_rc_mod;
            _INT4       *user_rc;
```

The Signal the Termination-Imminent Condition (CEETREC) API is used to do a normal ending of the activation group containing the nearest control boundary. First, the terminate-imminent condition is sent to give all intervening call stack entries a chance to clean up, or stop the end operation. If the resume cursor is not moved and a resume is requested, all call stack entries to the nearest control boundary end. If the call stack entry for the control boundary is also the oldest call stack entry in the activation group, the activation group ends.

Parameters

cel_rc_mod (input/optional)
A language-specific return code passed from one ILE language to another ILE language. The value and meaning is language-specific.

user_rc (input/optional)
A number representing the user portion of the activation group return code. If this parameter is not supplied, the CEETREC API uses the current contents of the activation group return code. If it is supplied, it takes precedence over previously set values.

Conditions

CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- A normal end operation by high-level language exit statements is implemented by calling the CEETREC API. This results in a termination-imminent condition.
- The activation group and job-level return codes are updated whether or not the activation group ended.
- This API cannot end the default activation group.

Unregister Call Stack Entry Termination User Exit Procedure (CEEUTX) API

Syntax

```
void CEEUTX (procedure, [fc])

            _RTX_ENTRY  *procedure;
            _FEEDBACK   *fc;
```

The Unregister Call Stack Entry Termination User Exit Procedure (CEEUTX) API is used to unregister a user-defined procedure that was previously registered by the Register Call Stack Entry Termination User Exit Procedure (CEERTX) API. The CEEUTX API operates on the user call stack entry termination exits that are registered for the call stack entry from which the CEEUTX API is called.

Parameters

procedure (input)
An entry variable or constant for the procedure that is to be unregistered for the call stack entry.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0252 Severity: 30	&1 is unable to find the procedure
CEE0253 Severity: 10	Additional registrations of the procedure remain in the queue
CEE0257 Severity: 30	The procedure provided for &1 is not valid
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- Registered call stack entry termination user exit procedures are automatically unregistered when the call stack entry for which they are registered no longer exists.
- If the same procedure is registered for the call stack entry more than once, the CEEUTX API processes the registrations in LIFO order.
- The message CEE0257 occurs if the pointer contained in *procedure* is not a procedure pointer.

Chapter 34. Condition Management APIs

ILE condition management APIs allow you to handle errors independent of high-level language-specific error handling.

The condition management APIs are:

- Construct a Condition Token (CEENCOD)
- Decompose a Condition Token (CEEDCOD)
- Handle a Condition (CEE4HC)
- Move the Resume Cursor to a Return Point (CEEMRCR)
- Register a User-Written Condition Handler (CEEHDLR)
- Retrieve ILE Version and Platform ID (CEEGPID)
- Return the Relative Invocation Number (CEE4RIN)
- Signal a Condition (CEESGL)
- Unregister a User Condition Handler (CEEHDLU)

The APIs are presented in alphabetical order.

How Conditions Are Represented

A condition token is used to communicate information about a condition to bindable APIs. It is also used to communicate information to the condition manager, to message services, and to procedures.

The ILE **condition token** is a 12-byte compound data type that contains structured fields to convey aspects of a condition. It conveys aspects such as the severity, the associated message number, and information that is specific to the given instance of the condition.

Condition Token Layout

Figure 34-1 displays a map of the condition token.

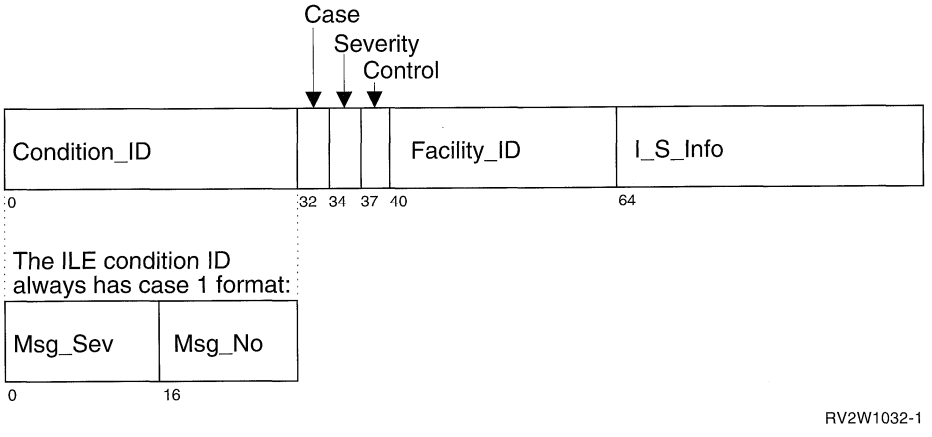


Figure 34-1. ILE Condition Token Layout

Every condition token contains the components indicated in Figure 34-1:

Condition_ID
: A 4-byte identifier that, with the Facility_ID, describes the condition that the token communicates. ILE bindable APIs and most applications produce Case 1 conditions.

Case
: A 2-bit field that defines the format of the Condition_ID portion of the token. ILE conditions are always Case 1.

Severity
: A 3-bit binary integer that indicates the condition's severity. Severity and Msg_Sev contain the same information. See Figure 34-2 on page 34-2 for a list of ILE condition severities. See Figure 34-3 on page 34-2 and Figure 34-4 on page 34-2 for the corresponding OS/400 message severities.

Control
: A 3-bit field containing flags that describe or control various aspects of condition handling. The third bit specifies whether the Facility_ID has been assigned by IBM.

Facility_ID
: A 3-character alphanumeric string that identifies the facility that generated the condition. Although all ILE languages use ILE message and condition handling bindable APIs, the actual run-time messages generated under ILE still carry the HLL identification in the Facility_ID.

I_S_Info
: A 4-byte field that identifies the instance specific information associated with a given instance of the condition. The instant specific information contains the reference key to the instance of the message associated with the condition token. If the message reference key is zero, there is no associated message.

Msg_Sev
: A 2-byte binary integer that indicates the condition's severity. Msg_Sev and Severity contain the same information. See Figure 34-2 on page 34-2 for a list of ILE

condition severities. See Figure 34-3 on page 34-2 and Figure 34-4 on page 34-2 for the corresponding OS/400 message severities.

Msg_No
A 2-byte, binary number that identifies the message associated with the condition. The combination of Facility_ID and Msg_No uniquely identifies a condition.

Figure 34-2 contains default responses that the condition manager takes when a handler attempts to percolate a function check across a control boundary.

Figure 34-2. Default Responses to Unhandled Exceptions

Message Type	Severity of Condition	Condition Raised by the Signal a Condition (CEESGL) Bindable API	Exception Originated from Any Other Source
Status	0 (Informative message)	Return the unhandled condition.	Resume without logging the message.
Status	1 (Warning)	Return the unhandled condition.	Resume without logging the message.
Notify	0 (Informative message)	Not applicable.	Log the notify message and send the default reply.
Notify	1 (Warning)	Not applicable.	Log the notify message and send the default reply.
Escape	2 (Error)	Return the unhandled condition.	Log the escape message and send a function check message to the call stack entry of the current resume point.
Escape	3 (Severe error)	Return the unhandled condition.	Log the escape message and send a function check message to the call stack entry of the current resume point.
Escape	4 (Critical ILE error)	Log the escape message and send a function check message to the call stack entry of the current resume point.	Log the escape message and send a function check message to the call stack entry of the current resume point.
Function check	4 (Critical ILE error)	Not applicable	End the application, and send the CEE9901 message to the caller of the control boundary.

Note: When the application is ended by an unhandled function check, the activation group is deleted if the control boundary is the oldest call stack entry in the activation group.

Figure 34-3 and Figure 34-4 show how ILE condition severity maps to OS/400 message severity.

*Figure 34-3. Mapping OS/400 *ESCAPE Message Severities to ILE Condition Severities*

From OS/400 Message Severity	To ILE Condition Severity	To OS/400 Message Severity
0-29	2	20
30-39	3	30
40-99	4	40

*Figure 34-4. Mapping OS/400 *STATUS and *NOTIFY Message Severities to ILE Condition Severities*

From OS/400 Message Severity	To ILE Condition Severity	To OS/400 Message Severity
0	0	0
1-99	1	10

Condition Token Testing

You can test a condition token that is returned from a bindable API for the following:

Success
To test for success, determine if the first 4 bytes are 0. If the first 4 bytes are 0, the remainder of the condition token is 0, indicating a successful call was made to the bindable API.

Equivalent Tokens
To determine whether two condition tokens are equivalent (that is, the same *type* of condition token, but not the same *instance* of the condition token), compare the first 8 bytes of each condition token with one another. These bytes are static and do not change depending upon the given instance when the condition occurs.

Equal Tokens
To determine whether two condition tokens are equal (that is, they represent the same *instance* of a condition), compare all 12 bytes of each condition token with one another. The last 4 bytes can change from instance to instance of a condition.

Construct a Condition Token (CEENCOD) API

Syntax

```
void CEENCOD (C_1, C_2, Case, Severity, Control,
              Facility_ID, I_S_Info, Cond_Token,
              [fc])

             _INT2       *C_1;
             _INT2       *C_2;
             _INT2       *Case;
             _INT2       *Severity;
             _INT2       *Control;
             _CHAR      (*Facility_ID)[3];
             _INT4       *I_S_Info;
             _FEEDBACK   *Cond_Token;
             _FEEDBACK   *fc;
```

The Construct a Condition Token (CEENCOD) API is used to dynamically construct a 12-byte condition token.

Parameters

C_1 (input)
A 2-byte binary integer representing the value of the first part of the Condition_ID. See Msg_Sev on page 34-1.

C_2 (input)
A 2-byte binary integer representing of the value of the second part of the Condition_ID. See the description of Msg_No on page 34-2.

Case (input)
A 2-byte binary integer that defines the format of the Condition_ID portion of the token. See the description of Case on page 34-1.

Severity (input)
A 2-byte unsigned binary integer that indicates the condition's severity. The value of this field is the same as in C_1.

Control (input)
A 2-byte binary number containing the control information of the condition. See the description of Control on page 34-1.

Facility_ID (input)
A 3-character field containing three alphanumeric characters that identify the product generating this condition or feedback information. See the description of Facility_ID on page 34-1.

I_S_Info
A 4-byte handle identifying the instance specific information. See the description of I_S_Info on page 34-1.

Cond_Token (output)
The 12-byte representation of the constructed condition or feedback information. See "How Conditions Are Represented" on page 34-1.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0401 Severity: 30	The Case parameter for &1 is not valid
CEE0402 Severity: 30	Unsupported control code &2 passed to procedure &1
CEE0403 Severity: 30	Severity passed to &1 is not valid

Usage Notes

- If the severity in C_1 does not match the severity in the Severity parameter, message CEE0403 is raised.

Decompose a Condition Token (CEEDCOD) API

Syntax

```
void CEEDCOD (Cond_Token, C_1, C_2, Case, Severity,
              Control, Facility_ID, I_S_Info, [fc])

             _FEEDBACK   *Cond_Token;
             _INT2       *C_1;
             _INT2       *C_2;
             _INT2       *Case;
             _INT2       *Severity;
             _INT2       *Control;
             _CHAR      (*Facility_ID)[3];
             _INT4       *I_S_Info;
             _FEEDBACK   *fc;
```

The Decompose a Condition Token (CEEDCOD) API returns the individual elements of an existing condition token.

Parameters

Cond_Token (input)
A condition token that represents the current condition or feedback information. See "How Conditions Are Represented" on page 34-1

C_1 (output)
A 2-byte binary integer representing the value of the first part of the Condition_ID. See the description of Msg_Sev on page 34-1.

C_2 (output)
A 2-byte binary integer representing of the value of the second part of the Condition_ID. See the description of Msg_No on page 34-2.

Case (output)
A 2-byte binary integer that defines the format of the Condition_ID portion of the token. See the description of Case on page 34-1.

Severity (output)
A 2-byte unsigned binary integer that indicates the severity of the condition. The value of this field is the same as in C_1.

Control (output)
A 16-bit field containing flags that describe aspects of the state of the condition. See the description of Control on page 34-1.

Facility_ID (output)
A 3-character field containing three alphanumeric characters that identify the product generating this condition or feedback information. See the description of Facility_ID on page 34-1.

I_S_Info
A 4-byte handle identifying the instance specific information. See the description of I_S_Info on page 34-1.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0102 Severity: 30	The condition token passed to &1 is not valid

Handle a Condition (CEE4HC) API

Syntax

```
void CEE4HC(isi, inv, [option], [fc])

            _UINT4      *isi;
            _INVPTR     *inv;
            _UINT4      *option;
            _FEEDBACK   *fc;
```

The Handle a Condition (CEE4HC) API handles a specified condition and, optionally, logs the condition.

Parameters

isi (input)
The instance specific information number for the condition. This is the message reference key of the underlying AS/400 exception.

inv (input)
The call stack entry that the handle cursor points to for the condition. This is the target call stack entry of the underlying AS/400 exception.

option (input/optional)
The options are:

0 Take the default log action for the message. *ESCAPE and function check messages are logged. *STATUS messages are never logged.
1 Do not log the condition.
2 Log the condition. Only conditions with a severity greater than 1 can be logged.

If the option parameter is omitted, the value 0 is used.

fc(output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE3104 Severity: 30	The pointer type in &1 is not valid
CEE3105 Severity: 30	The call stack entry given to &1 no longer exists
CEE3107 Severity: 30	The message specified for &1 is not an exception message
CEE3108 Severity: 30	The option specified is not valid
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- The call stack entry that *inv* points to must appear in the call stack for the job issuing the call to the CEE4HC API.
- The CEE4HC API can be used to take a previously handled message out of the job log, but only if that message is enqueued on *inv*.
- The CEE4HC API can be used to handle *ESCAPE, *STATUS, *NOTIFY, and function check exceptions. A default reply is sent for *NOTIFY messages if no reply has been sent previously.

Move the Resume Cursor to a Return Point (CEEMRCR) API

Syntax

```
void CEEMRCR (type_of_move, [fc])

              _INT4       *type_of_move;
              _FEEDBACK   *fc;
```

The Move the Resume Cursor to a Return Point (CEEMRCR) API moves the resume cursor to a return point relative to the current handle cursor.

Initially, the resume cursor is positioned after the machine instruction that caused the condition to be raised. The direction of movement is always toward older call stack entries.

Parameters

type_of_move (input)
The type of movement of the resume cursor relative to the current position of the handle cursor. The values are:

0 Move the resume cursor from a later call stack entry to the call stack entry that is currently associated with the handle cursor.

A handler moves the resume cursor if resumption in the call stack entry at which the resume cursor is pointing is not possible. Resumption may be possible in the call stack entry at which the handle cursor is pointing.

1 Move the resume cursor to the *call return* point (immediately following the *call* statement) for the call stack entry one prior to the position of the handle cursor. Also move the handle cursor to the first handler of the call stack entry the resume cursor is being moved to. This action exits the current call stack entry and skips all condition handlers still to be called for the call stack entry.

The handler moves the resume cursor if resumption is impossible at the current location, but may be possible in the caller of the current procedure.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0254 Severity: 10	The type of move for &1 is not valid
CEE0260 Severity: 30	No active condition for call to &1
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- The actual movement occurs only after the condition handler returns to the condition manager. Multiple calls to this API yield the net result of the calls. That is, if two calls from a given user-written condition handler set the resume cursor to different places, the most recent call will be used.
- When a return operation is made to the condition manager after the resume cursor is moved, any associated exit procedures are called as each call stack entry is passed. Moving a resume cursor past a call stack entry also cancels any associated user-written condition handlers.
- Ensure that the CEEMRCR API is called before handling the condition. A call to the CEEMRCR API is not valid if there is no active condition. For example, the condition may not be active if the condition handler uses the message handler API, Change Exception Message (QMHCHGEM), to handle the condition.

Register a User-Written Condition Handler (CEEHDLR) API

Syntax

```
void CEEHDLR (procedure, token, [fc])

              _HDLR_ENTRY *procedure;
              _POINTER    *token;
              _FEEDBACK   *fc;
```

The Register a User-Written Condition Handler (CEEHDLR) API registers a user-written condition handler for the current call stack entry.

Parameters

procedure (input)
An entry variable or constant for the procedure that is to be called to process the conditions.

token (input)
A pointer passed to the user condition handler each time the pointer is called.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0256 Severity: 10	Procedure already registered, registered again
CEE0257 Severity: 30	The procedure provided for &1 is not valid
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- A queue of handlers is maintained for each call stack entry. The handlers are given control in LIFO order. That is, the handler most recently registered is the first one used for the call stack entry from which the call to CEEHDLR was made.
- Any registered user-written condition handlers that were not unregistered by the Unregister a User Condition

Handler (CEEHDLU) API, are unregistered automatically by ILE upon removal of the associated call stack entry from the call stack. For more information on unregistering user-written condition handlers, see "Unregister a User Condition Handler (CEEHDLU) API" on page 34-8.

- The message CEE0257 occurs if the pointer contained in *procedure* is not a procedure pointer.

ILE Condition Handler Interface

Following is a description of the interface that the system uses to communicate with ILE condition handlers.

Syntax

```
void condition_handler(C_CTOK, token, result_code,
                       new_condition)

            _FEEDBACK   *C_CTOK;
            _POINTER    *token;
            _INT4       *result_code;
            _FEEDBACK   *new_condition;
```

Parameters

C_CTOK(input)
Identifies the current condition being processed.

token (input)
The token that was passed to the system with the call to CEEHDLR that registered this condition handler.

result_code (output)
This field contains the instructions from the condition handler to the system regarding the actions that the system should take.

ILE condition handlers get control for all *ESCAPE, *STATUS, *NOTIFY, and function check messages. Not all result code actions are valid for all types of messages.

If the message is handled by the ILE condition handler, the result-code action is not performed.

If a result code is returned that is not valid, the following message occurs:

CEE0265 Severity:30	The result code received from a condition handler is not valid

Valid result codes are:

Resume
This result code can be used for all exception types.

10 Resume at the resume cursor, and handle the condition, as follows:
Function Check (severity 4)
The message appears in the job log.
***ESCAPE (severity 2-4)**
The message appears in the job log.
***STATUS (severity 1)**
The message does not appear in the job log.
***NOTIFY**
The default reply is sent and the message appears in the job log.

Percolate
These result codes can be used for all exception types.

20 Percolate to the next condition handler.
21 Percolate to the next call stack entry. This can skip a high-level language condition handler for this call stack entry. Any remaining user handlers in the queue for this call stack entry also can be skipped.

This handle cursor movement is in addition to any other handle cursor movement done explicitly in the handler, for example by the CEEMRCR API.

The handle cursor is not moved past a call stack entry for a control boundary. The default condition handling actions are applied to the message. Figure 34-2 on page 34-2 summarizes the default condition handling actions.

Promote
Only *ESCAPE and *STATUS messages may be promoted.

30 Promote to the next condition handler.
31 Promote to the next call stack entry. This may skip a high-level language condition handler for this call stack entry. Any remaining user handlers in the queue for this call stack entry also can be skipped.

This handle cursor movement is in addition to any other handle cursor movement done explicitly in the handler, for example by the CEEMRCR API.

The handle cursor is not moved past a call stack entry for a control boundary. The default condition handling actions are applied to the new message. Figure 34-2 on page 34-2 summarizes the default condition handling actions.
32 Promote and restart condition handling with the first condition handler for the call stack entry at which the handle cursor currently points.

Note: It is not valid to promote a condition without returning a new condition token. If the original condition is returned in *new_condition*, the following message occurs:

CEE0262 Severity:30	The condition being promoted is not valid

new_condition (output)
The condition token representing the promoted condition. This field is used only for *result_code* values of 30, 31, 32, and 60 denoting *promote* or *fix-up and resume*.

Retrieve ILE Version and Platform ID (CEEGPID) API

Syntax

```
void CEEGPID (CEE_Version, Plat_ID, [fc])

            _INT4       *CEE_Version;
            _INT4       *Plat_ID;
            _FEEDBACK   *fc;
```

The Retrieve ILE Version and Platform ID (CEEGPID) API retrieves the ILE version ID and the platform (that is, operating system) ID. The IDs are those currently in use for processing the active condition.

Parameters

CEE_Version(output)
A 32-bit numeric representation of the version of ILE that created this condition. For example, if 230 is returned as the version, it represents Version 2 Release 3 Modification 0.

Plat_ID(output)
A 32-bit numeric representation of the operating system on which this condition was created. The possible values are:

2 OS/2
3 MVS/VM/370
4 OS/400

fc(output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE9902 Severity: 30	Unexpected user error occurred in &1

Return the Relative Invocation Number (CEE4RIN) API

Syntax

```
void CEE4RIN (rel_inv, inv, [fc])

            _INT4       *rel_inv;
            _INVPTR     *inv;
            _FEEDBACK   *fc;
```

The Return the Relative Invocation Number (CEE4RIN) API retrieves the relative invocation number of an invocation pointer, and returns it in *rel_inv*.

Parameters

rel_inv (output)
The relative invocation number. This is a positive number indicating the number of invocations back from the caller that *inv* occurs.

inv (input)
An invocation pointer.

fc(output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE3104 Severity: 30	The pointer type in &1 is not valid
CEE3105 Severity: 30	The call stack entry given to &1 no longer exists
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- The invocation specified by *inv* must appear in the call stack for the job issuing the CEE4RIN API.

Signal a Condition (CEESGL) API

Syntax

```
void CEESGL (cond_rep, [Q_Data_Token], [fc])

            _FEEDBACK   *cond_rep;
            _INT4       *q_data_token;
            _FEEDBACK   *fc;
```

The Signal a Condition (CEESGL) API signals or resignals a condition to the ILE condition manager.

Parameters

cond_rep (input/output)
A condition token defining the condition to be raised.

The CEESGL API always uses the facility_ID to retrieve a message, whether or not instance specific information (ISI) is provided. If ISI is provided, the message data from the message is used as insert data for the condition that is to be raised.

q_data_token (input/optional)
A 32-bit data object to be placed in the ISI field for use in accessing the qualifying data associated with the given instance of the condition. This parameter is provided for compatibility purposes only. It is ignored on the AS/400 system.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0201 Severity: 00	The condition sent by &1 is not handled
CEE0258 Severity: 30	The condition token passed to &1 is not valid
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- If the condition is unhandled and a feedback code is provided, a message number is returned in the fc parameter. The message number returned is dependent on the severity of the condition when it reaches the control boundary.

  If the severity of the condition is 0-1, message number CEE0000 is returned. The message is classified as a *STATUS message.

  If the severity of the condition is 2-4, message number CEE0201 is returned. The message is classified as an *ESCAPE message.

  See Figure 34-3 on page 34-2 and Figure 34-4 on page 34-2 for the relationship of condition severities and message severities.

Unregister a User Condition Handler (CEEHDLU) API

Syntax

```
void CEEHDLU (procedure, [fc])

              _HDLR_ENTRY *procedure;
              _FEEDBACK   *fc;
```

The Unregister a User Condition Handler (CEEHDLU) API unregisters a user-written condition handler for the current call stack entry.

Parameters

procedure (input)
An entry variable or constant for a user-written condition handler that was previously registered for the current call stack entry using the Register a User-Written Condition Handler (CEEHDLR) API. For more information on registering user-written condition handlers, see "Register a User-Written Condition Handler (CEEHDLR) API" on page 34-5.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0252 Severity: 10	&1 is unable to find the procedure
CEE0253 Severity: 10	Additional registrations of the procedure remain in the queue
CEE0257 Severity: 30	The procedure provided for &1 is not valid
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- Any registered user-written condition handlers not unregistered by CEEHDLU, are unregistered automatically by the system upon removal of the associated call stack entry from the call stack.
- If the specified procedure is registered more than once, the most recent registration is removed. Earlier registrations remain in the queue for the call stack entry.
- The message CEE0257 occurs if the pointer contained in *procedure* is not a procedure pointer.

Chapter 35. Date and Time APIs

The date and time APIs are:

- Calculate Day of Week from Lilian Date (CEEDYWK)
- Convert Date to Lilian Format (CEEDAYS)
- Convert Integers to Seconds (CEEISEC)
- Convert Lilian Date to Character Format (CEEDATE)
- Convert Seconds to Character Timestamp (CEEDATM)
- Convert Seconds to Integers (CEESECI)
- Convert Timestamp to Number of Seconds (CEESECS)
- Get Current Greenwich Mean Time (CEEGMT)
- Get Current Local Time (CEELOCT)
- Get Offset from Universal Time Coordinated to Local Time (CEEUTCO)
- Get Universal Time Coordinated (CEEUTC)
- Query Century (CEEQCEN)
- Return Default Date and Time Strings for Country (CEEFMDT)
- Return Default Date String for Country (CEEFMDA)
- Return Default Time String for Country (CEEFMTM)
- Set Century (CEESCEN)

The APIs are presented in alphabetical order.

Date and Time Notation and Limits

Calendars used by the date and time bindable APIs have specific notation and limits associated with them. Following is a description of the notation and the limits.

Notation

Calendars based on eras[1] use unique strings to identify these eras. These identification strings are called picture strings. The following picture strings are supported:

Japanese era Identified by one 6-character string <JJJJ>

Republic of China (ROC) era
Identified by one 6-character string <CCCC>, or by one 10-character string <CCCCCCCC>.

The era picture string begins with the less than character (<) and ends with the greater than character (>). The characters within these characters are either uppercase Js or uppercase Cs.

Limits

Date variables associated with certain calendars have certain limitations. These limits are:

starting Lilian date
The beginning of the Lilian date range is set to 15 October 1582. This is defined as Lilian day 1. Lilian day zero is defined as 14 October 1582, and is used for calculation purposes in several of the APIs. Lilian second 1 is defined as 00:00:01 on 14 October 1582, and is used for calculation purposes in several of the APIs. Only Lilian dates greater than or equal to 1 are valid as input or produced as output in the date and time APIs. This is the date of adoption of the Gregorian calendar. Lilian dates preceding this date are undefined.

end Lilian date
The end of the Lilian date range is set to 31 December 9999. Lilian dates following this date are undefined.

limit of current era
The maximum future date that can be expressed in the era system must be within the first 999 years of the current era. Future dates beyond year 999 of the current era are not defined.

number of eras
The eras supported by ILE are shown within this section. No other past eras are supported by ILE at the present time. Future eras will be added as required.

Calculate Day of Week from Lilian Date (CEEDYWK) API

Syntax

```
void CEEDYWK (input_Lilian_date, output_day_no, [fc])

              _INT4       *input_Lilian_date;
              _INT4       *output_day_no;
              _FEEDBACK   *fc;
```

The Calculate Day of Week from Lilian Date (CEEDYWK) API returns the day of the week as a number between 1 and 7.

Parameters

input_Lilian_date (input)
A 32-bit binary integer representing the Lilian date, which is the number of days since 14 October 1582. For example, 16 May 1988 is day number 148 138. The valid range is 1 to 3 074 324 (31 December 9999).

output_day_no (output)
A 32-bit binary integer representing the day of week of the *input_Lilian_date*. For the day of week, 1 indicates Sunday, 2 indicates Monday, ..., 7 indicates Saturday. If

[1] An era is defined by a major event in time from which date calculations take place.

input_Lilian_date is not valid, *output_day_no* is set to 0 and CEEDYWK ends with a nonzero feedback code.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE2512 Severity: 30	The value for the given Lilian date is not valid

Usage Notes

- The number returned by CEEDYWK is useful for end-of-week calculations.

Convert Date to Lilian Format (CEEDAYS) API

Syntax

```
void CEEDAYS (input_char_date, picture_string,
              output_Lilian_date, [fc])

            _CHAR       *input_char_date;
            _CHAR       *picture_string;
            _INT4       *output_Lilian_date;
            _FEEDBACK   *fc;
```

The Convert Date to Lilian Format (CEEDAYS) API converts a string representing a date into a number representing the number of days since 14 October 1582. This API makes it easier to do calculations such as the number of days between two dates.

Parameters

input_char_date (input by descriptor)
A character string representing a date or timestamp in the format shown by *picture_string*. Field width is 5 to 255 characters. *Input-char-date* can contain leading or trailing blanks. Parsing for a date begins with the first non-blank character unless the picture string contains leading blanks, in which case CEEDAYS skips exactly that many positions before parsing begins. After a valid date is parsed, remaining characters are ignored. Valid dates are in the range 15 October 1582 to 31 December 9999.

picture_string (input by descriptor)
A character string indicating the format of the date value in *input_char_date*, for example `MM/DD/YY`. Each character in *picture_string* represents a character in *input_char_date*. If delimiters such as the slash (/) appear in the picture string, then leading zeros can be omitted. For example:

```
CALL CEEDAYS('6/2/88'  , 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('06/02/88', 'MM/DD/YY', lildate, fc);
CALL CEEDAYS('060288'  , 'MMDDYY'  , lildate, fc);
CALL CEEDAYS('88154'   , 'YYDDD'   , lildate, fc);
```

would all assign the same value to variable `lildate`. If any time characters are included, for example `HH:MI:SS YY/MM/DD`, they count as place holders but are otherwise ignored.

Figure 35-1 on page 35-3 contains a list of valid picture characters, and Figure 35-2 on page 35-4 has examples of valid picture strings.

If *picture_string* is null or blank, CEEDAYS obtains *picture_string* based on the current job value for COUNTRY ID. For example, if the current value for COUNTRY ID is US (United States), the date format is `MM/DD/YYYY`. If the current job value for COUNTRY ID is FR (France), the date format is `DD.MM.YYYY`.

This default mechanism makes it easy for translators to specify the preferred date format, and also easy for application programs and library procedures to automatically use this format.

output_Lilian_date (output)
A 32-bit binary integer representing the Lilian date, which is the number of days since 14 October 1582. For example, 16 May 1988 is day number 148138. If *input_char_date* does not contain a valid date, *output_Lilian_date* is set to 0 and CEEDAYS ends with a nonzero feedback code.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0501 Severity: 30	The operational descriptor data type is not valid
CEE0502 Severity: 30	Missing operational descriptor
CEE2507 Severity: 30	Insufficient data provided
CEE2508 Severity: 30	The value for day is not valid
CEE2509 Severity: 30	The value for era is not valid

CEE2513 Severity: 30	The value for Lilian date is not valid
CEE2517 Severity: 30	The value for month is not valid
CEE2518 Severity: 30	The picture string specification is not valid
CEE2520 Severity: 30	The picture string is not valid
CEE2521 Severity: 30	The value for year is not valid

Usage Notes

- The inverse of CEEDAYS is CEEDATE. The CEEDATE API converts a date in the Lilian format to character format.
- Date calculations can be performed easily on the *output_Lilian_date*, because it is an integer. Leap year and end-of-year anomalies are avoided.
- See "Set Century (CEESCEN) API" on page 35-16 and "Query Century (CEEQCEN) API" on page 35-14 for information on 2-digit years.
- If picture_string includes a Japanese era symbol <JJJJ>, the YY position in *input_char_date* is assumed to contain the year within Japanese era. Figure 35-2 on page 35-4 has an example. Figure 35-3 on page 35-5 contains a list of the Japanese eras recognized by CEEDAYS.
- If *picture_string* includes a Republic of China (ROC) era symbol <CCCC> or <CCCCCCCC>, the YY position in *input_char_date* is assumed to contain the year within ROC era. Figure 35-2 on page 35-4 has an example. Figure 35-4 on page 35-5 contains a list of the ROC eras recognized by CEEDAYS. The coding of YYY or ZYY without including one of the era symbols is a picture string specification error.

Figure 35-1 (Page 1 of 2). Picture Characters Used in Picture Strings

Picture Characters	Explanation	Valid Values	Notes
Y YY YYY ZYY YYYY	1-digit year 2-digit year 3-digit year 3-digit year within era 4-digit year	0-9 00-99 000-999 1-999 1582-9999	Y valid for output only. YY implies the years xx00-xx99. (The years are dependent on the century start value and the system date.) YYY or ZYY used with <JJJJ>, <CCCC>, and <CCCCCCCC>.
<JJJJ>	Japanese era name in DBCS characters	*Heisei* (X'0E458D45BA0F') *Showa* (X'0E45B3457A0F') *Taisho* (X'0E455B45770F') *Meiji* (X'0E45A645840F')	Affects YY field: if <JJJJ> specified, YY means the year within Japanese era. For example, 1988 = Showa 63. See example in Figure 35-2 on page 35-4.
<CCCC> <CCCCCCCC>	Republic of China (ROC) era name in DBCS characters	*MinKow* (X'0E4D8256CE0F') *ChuHwaMinKow* (X'0E4C845ADD4D8256CE0F')	Affects YY field: if <CCCC> specified, YY means the year within ROC era. For example, 1988 = Minkow 77. See example in Figure 35-2 on page 35-4.
MM ZM	2-digit month 1- or 2-digit month	01-12 1-12	For output, leading zero suppressed. For input, ZM treated as MM.
RRRR RRRZ	Roman numeral month	I-XII (Left-justified)	For input, source string is folded to uppercase. For output, uppercase only. I=Jan, II=Feb, ..., XII=Dec.
MMM Mmm MMMMMMMMM Mmmmmmmmm MMMMMMMMZ Mmmmmmmmz	3-char month, uppercase 3-char month, mixed case 20-char month, uppercase 20-char month, mixed case trailing blanks suppressed trailing blanks suppressed	JAN-DEC Jan-Dec JANUARYbb-DECEMBERb Januarybb-Decemberb JANUARY-DECEMBER January-December	For input, source string always folded to uppercase. For output, M generates uppercase and m generates lowercase. Output is padded with blanks (b) (unless Z specified) or truncated to match the number of Ms.
DD ZD DDD	2-digit day of month 1- or 2-digit day of month day of year (Julian day)	01-31 1-31 001-366	For output, leading zero suppressed. For input, ZM and MM are equivalent. That is, each accepts the format of the other.

Figure 35-1 (Page 2 of 2). Picture Characters Used in Picture Strings

Picture Characters	Explanation	Valid Values	Notes
HH ZH	2-digit hour 1- or 2-digit hour	00-23 0-23	For output, leading zero suppressed. For input, ZH and HH are equivalent. That is, each accepts the format of the other. If AP is specified, valid values are 01-12.
MI	minute	00-59	
SS	second	00-59	
9 99 999	tenths of a second hundredths of a second thousandths of a second	0-9 00-99 000-999	No rounding.
AP ap A.P. a.p.	AM/PM indicator	AM or PM am or pm A.M. or P.M.	AP affects HH/ZH field. For input, source string always folded to uppercase. For output, AP generates uppercase and ap generates lowercase.
W WWW Www WWWWWWWWW Wwwwwwwww WWWWWWWWWZ Wwwwwwwwwz	1-char day-of-week 3-char day, uppercase 3-char day, mixed case 10-char day, uppercase 10-char day, mixed case trailing blanks suppressed trailing blanks suppressed	S, M, T, W, T, F, S SUN-SAT Sun-Sat SUNDAYbbb-SATURDAYb Sundaybbb-Saturdayb SUNDAY-SATURDAY Sunday-Saturday	For input, Ws are ignored. For output, W generates uppercase and w generates lowercase. Output padded with blanks (unless Z specified) or truncated to match the number of Ws.
All others, not including numbers 0 through 9.	delimiters	X'01'-X'FF' (X'00' reserved for ILE use.)	For input, treated as delimiters between the month, day, year, hour, minute, second, and fraction of a second. For output, copied exactly as is to the target string.

Figure 35-2 (Page 1 of 2). Examples of Picture Strings Recognized by ILE Date and Time APIs

Picture String	Example	Notes
YYMMDD	880516	
YYYYMMDD	19880516	
YYYY-MM-DD	1988-05-16	1988-5-16 would also be valid input.
<JJJJ> YY.MM.DD	*Showa* 63.05.16	*Showa* is a Japanese era name. *Showa* 63 = 1988.
<CCCC> YY.MM.DD	*MinKow* 77.05.16	*MinKow* is an ROC era name. *MinKow* 77 = 1988.
MMDDYY	050688	
MM/DD/YY	05/06/88	
ZM/ZD/YY	5/6/88	
MM/DD/YYYY	05/06/1988	
MM/DD/Y	05/06/8	1-digit year format (Y) valid for output only
DD.MM.YY	09.06.88	
DD-RRRR-YY	09- VI-88	
DD MMM YY	09 JUN 88	
DD Mmmmmmmmm YY	09 June 88	
ZD Mmmmmmmmz YY	9 June 88	Z suppresses zeros and blanks
Mmmmmmmmz ZD, YYYY	June 9, 1988	
ZDMMMMMMMMZYY	9JUNE88	

Figure 35-2 (Page 2 of 2). Examples of Picture Strings Recognized by ILE Date and Time APIs

Picture String	Example	Notes
YY.DDD YYDDD YYYY/DDD	88.137 88137 1988/137	Julian date
YYMMDDHHMISS YYYYMMDDHHMISS YYYY-MM-DD HH:MI:SS.999 WWW, ZM/ZD/YY HH:MI AP Wwwwwwwwwz, DD Mmm YYYY, ZH:MI AP	880516204229 19880516204229 1988-05-16 20:42:29.046 MON, 5/16/88 08:42 PM Monday, 16 May 1988, 8:42 PM	Timestamp — valid only for CEESECS and CEEDATM. If used with CEEDATE, time positions are left blank. If used with CEEDAYS, HH, MI, SS, and 999 fields are ignored.

Figure 35-3. Japanese Eras Used by ILE Date and Time APIs When <JJJJ> Specified

First date of Japanese Era	Era Name	Era Name in IBM Japanese DBCS Code	Valid Year (YY, ZYY) Values
1868-09-08	Meiji	X'0E45A645840F'	01-45
1912-07-30	Taisho	X'0E455B45770F'	01-15
1926-12-25	Showa	X'0E45B3457A0F'	01-64
1989-01-08	Heisei	X'0E458D45BA0F'	01-999 (01 = 1989)

Note: The Japanese era table is stored in the QSYS/QLEERA_JP data area.

Figure 35-4. Republic of China Eras Used by ILE Date and Time APIs When <CCCC> or <CCCCCCCC> Specified

First date of ROC Era	Era Name	Era Name in Traditional Chinese DBCS Code	Valid Year (YY, ZYY) Values
1912-01-01	*MinKow*	X'0E4D8256CE0F'	01-999 (77 = 1988)
	ChuHwaMinKow	X'0E4C845ADD4D8256CE0F'	

Note: The Chinese era table is stored in the QSYS/QLEERA_TW data area.

Example

- Convert a date to the Lilian format to calculate the date 60 days hence:

```
CALL CEEDAYS ('880516','YYMMDD', ndays, fc);
ndays = ndays + 60;
CALL CEEDATE (ndays, 'YYMMDD', newdate, fc);
```

Convert Integers to Seconds (CEEISEC) API

Syntax

```
void CEEISEC (input_year, input_month, input_day,
              input_hours, input_minutes,
              input_seconds, input_milliseconds,
              output_seconds, [fc])

             _INT4       *input_year;
             _INT4       *input_month;
             _INT4       *input_day;
             _INT4       *input_hours;
             _INT4       *input_minutes;
             _INT4       *input_seconds;
             _INT4       *input_milliseconds;
             _FLOAT8     *output_seconds;
             _FEEDBACK   *fc;
```

The Convert Integers to Seconds (CEEISEC) API converts separate binary integers representing year, month, day, hour, minute, second, and millisecond to a number representing the number of seconds since 00:00:00 14 October 1582. Use CEEISEC instead of CEESECS when the input is in numeric format rather than character format.

Parameters

input_year (input)
A 32-bit binary integer representing year. The range is 1582 through 9999.

input_month (input)
A 32-bit binary integer representing month. The range is 1 through 12.

input_day (input)
A 32-bit binary integer representing day. The range is 1 through 31.

input_hours (input)
A 32-bit binary integer representing hours. The range is 0 through 23.

input_minutes (input)
A 32-bit binary integer representing minutes. The range is 0 through 59.

input_seconds (input)
A 32-bit binary integer representing seconds. The range is 0 through 59.

input_milliseconds (input)
A 32-bit binary integer representing milliseconds. The range is 0 through 999.

output_seconds (output)
A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The range is 86,400 to 265 621 679 999.999 (23:59:59.999 31 December 9999). If any input values are not valid, *output_seconds* is set to zero.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE2510 Severity: 30	The value for hour is not valid
CEE2511 Severity: 30	The value for day is not valid
CEE2513 Severity: 30	The value for Lilian date is not valid
CEE2514 Severity: 30	The value for Lilian year is not valid
CEE2515 Severity: 30	The value for millisecond is not valid
CEE2516 Severity: 30	The value for minute is not valid
CEE2517 Severity: 30	The value for month is not valid
CEE2519 Severity: 30	The value for second is not valid

Usage Notes

- The inverse of CEEISEC is CEESECI. The CEESECI API converts number of seconds to integer year, month, day, and so forth.
- To convert *output_seconds* to a Lilian day number, divide *output_seconds* by 86 400 (number of seconds in a day).
- CEEISEC can be used to do date arithmetic that cannot otherwise be done with Lilian dates or number of seconds. For example, to add exactly 6 months to a date rather than add 180 days, use CEEISEC.

Convert Lilian Date to Character Format (CEEDATE) API

Syntax

```
void CEEDATE (input_Lilian_date, picture_string,
              output_char_date, [fc])

             _INT4       *input_Lilian_date;
             _CHAR       *picture_string;
             _CHAR       *output_char_date;
             _FEEDBACK   *fc;
```

The Convert Lilian Date to Character Format (CEEDATE) API formats a number representing a Lilian date. The output is a character string such as 1988/07/26.

Parameters

input_Lilian_date (input)
A 32-bit binary integer representing the Lilian date, which is the number of days since 14 October 1582. For example, 16 May 1988 is day number 148 138. Valid range is 1 to 3 074 324 (31 December 9999).

picture_string (input by descriptor)
A character string representing the desired format of *output_char_date*, for example `MM/DD/YY`. Each character in *picture_string* represents a character in *output_char_date*. If delimiters such as the slash (/) appear in the picture string, they are copied as is to *output_char_date*.

Figure 35-1 on page 35-3 has a list of valid picture characters, and Figure 35-2 on page 35-4 contains examples of valid picture strings.

If *picture_string* is null or blank, CEEDATE obtains *picture_string* based on the current job value for COUNTRY ID. For example, if the current job value for COUNTRY ID is US (United States), the date format is

MM/DD/YYYY. If the current job value for COUNTRY ID is FR (France), the date format is DD.MM.YYYY.

This default mechanism makes it easy for translators to specify the preferred date format, and also easy for application programs and library procedures to automatically use this format.

output_char_date (output by descriptor)
A character string that is the result of converting *input_Lilian_date* to the format specified by *picture_string*. If necessary, output will be truncated to the length of *output_char_date*. Figure 35-5 contains sample output dates. If *input_Lilian_date* is not valid, *output_char_date* is set to all blanks and CEEDATE ends with a nonzero *feedback-code*.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0501 Severity: 30	The operational descriptor data type is not valid
CEE0502 Severity: 30	Missing operational descriptor
CEE2512 Severity: 30	The value for the given Lilian date is not valid
CEE2518 Severity: 30	The picture string specification is not valid
CEE2522 Severity: 30	Lilian date outside of era
CEE2526 Severity: 30	Date truncated

Usage Notes

- The inverse of CEEDATE is CEEDAYS. The CEEDAYS API converts character dates to the Lilian format.
- If *picture_string* includes a Japanese era symbol <JJJJ>, the YY position in *output_char_date* is replaced by "year within Japanese era." Figure 35-2 on page 35-4 has an example. Figure 35-3 on page 35-5 contains a list of Japanese eras supported by CEEDATE.
- If *picture_string* includes a Republic of China (ROC) Era symbol <CCCC> or <CCCCCCCC>, the YY position in *output_char_date* is replaced by the year within ROC era. Figure 35-2 on page 35-4 has an example. Figure 35-4 on page 35-5 contains a list of ROC eras supported by CEEDATE.

Figure 35-5 (Page 1 of 2). Sample Output of the CEEDATE API

input_Lilian_date	*picture_string*	*output_char_date*
148138	YY	88
	YYMM	8805
	YY-MM	88-05
	YYMMDD	880516
	YYYYMMDD	19880516
	YYYY-MM-DD	1988-05-16
	YYYY-ZM-ZD	1988-5-16
	<JJJJ> YY.MM.DD	*Showa* 63.05.16 (in a DBCS string)
	<CCCC> YY.MM.DD	*MinKow* 77.05.16 (in a DBCS string)
148139	MM	05
	MMDD	0517
	MM/DD	05/17
	MMDDYY	051788
	MM/DD/YYYY	05/17/1988
	ZM/DD/YYYY	5/17/1988
148140	DD	18
	DDMM	1805
	DDMMYY	180588
	DD.MM.YY	18.05.88
	DD.MM.YYYY	18.05.1988
	DD Mmm YYYY	18 May 1988
148141	DDD	140
	YYDDD	88140
	YY.DDD	88.140
	YYYY.DDD	1988.140

Figure 35-5 (Page 2 of 2). Sample Output of the CEEDATE API

input_Lilian_date	*picture_string*	*output_char_date*
148142	YY/MM/DD HH:MI:SS.99 YYYY/ZM/ZD ZH:MI AP	88/05/20 00:00:00.00 88/5/20 0:00 AM
148143	WWW., MMM DD, YYYY Www., Mmm DD, YYYY Wwwwwwwwww, Mmmmmmmmm DD, YYYY Wwwwwwwwwz, Mmmmmmmmmz DD, YYYY	SAT., MAY 21, 1988 Sat., May 21, 1988 Saturdaybb, Maybbbbbbb 21, 1988 Saturday, May 21, 1988

Example

- Convert a date from the Lilian format to IBM USA standard format MM/DD/YYYY:

```
CALL CEEDATE (lildate, 'MM/DD/YYYY', usadate, fc);
```

Convert Seconds to Character Timestamp (CEEDATM) API

Syntax

```
void CEEDATM (input_seconds, picture_string,
              output_timestamp, [fc])

             _FLOAT8     *input_seconds;
             _CHAR       *picture_string;
             _CHAR       *output_timestamp;
             _FEEDBACK   *fc;
```

The Convert Seconds to Character Timestamp (CEEDATM) API formats a number representing the number of seconds since 00:00:00 14 October 1582. The output is a character string such as 1988/07/26 20:37:00.

Parameters

input_seconds (input)
A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86,401 (24*60*60 + 01). The valid range is 86 400 to 265 621 679 999.999 (23:59:59.999 31 December 9999).

picture_string (input by descriptor)
A character string representing the desired format of *output_timestamp*, for example MM/DD/YY HH:MM AP. Each character in *picture_string* represents a character in *output_timestamp*. If delimiters such as the slash (/) appear in the picture string, they are copied as is to *output_char_date*.

Figure 35-1 on page 35-3 contains a list of valid picture characters, and Figure 35-2 on page 35-4 has examples of valid picture strings.

If *picture_string* is null or blank, CEEDATM obtains *picture_string* based on the current job value for COUNTRY ID. For example, if the current job value for COUNTRY ID is US (United States), the date-time format is MM/DD/YYYY HH:MI AP. If the current job value for COUNTRY ID is FR (France), the date-time format is YYYY-MM-DD HH.MI.

This default mechanism makes it easy for translators to specify the preferred timestamp format, and also easy for application programs and library procedures to automatically use this format.

output_timestamp (output by descriptor)
A character string that is the result of converting *input_seconds* to the format specified by *picture_string*. If necessary, output is truncated to the length of *output_timestamp*. Figure 35-6 on page 35-9 shows sample output. If input_seconds is not valid, *output_timestamp* is set to all blanks and CEEDATM ends with a nonzero *feedback-code*.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0501 Severity: 30	The operational descriptor data type is not valid
CEE0502 Severity: 30	Missing operational descriptor
CEE2505 Severity: 30	The value for seconds is out of range
CEE2506 Severity: 30	The value for seconds is outside of era
CEE2518 Severity: 30	The picture string specification is not valid
CEE2527 Severity: 20	Timestamp truncated

Usage Notes

- The inverse of CEEDATM is CEESECS. The CEESECS converts timestamp values to number-of-seconds values.
- If the input value is a Lilian date instead of seconds, multiply the Lilian date by 86400 (number of seconds in a day), and pass the new value to CEEDATM.
- If *picture_string* includes the Japanese era symbol <JJJJ>, the YY position in *output_timestamp* is replaced by the year within Japanese era. Figure 35-2 on page 35-4 has an example. Figure 35-3 on page 35-5 contains a list of Japanese eras supported by CEEDATM.
- If *picture_string* includes the Republic of China (ROC) era symbol <CCCC> or <CCCCCCCC>, the YY position in *output_timestamp* is replaced by the year within ROC era. See Figure 35-2 on page 35-4 for an example. See Figure 35-4 on page 35-5 for a list of ROC eras supported by CEEDATM.

Figure 35-6. Sample Output of the CEEDATM API

input_seconds	*picture_string*	*output_timestamp*
12799191601.000	YYMMDD	880516
	HH:MI:SS	19:00:01
	YY-MM-DD	88-05-16
	YYMMDDHHMISS	880516190001
	YY-MM-DD HH:MI:SS	88-05-16 19:00:01
	YYYY-MM-DD HH:MI:SS AP	1988-05-16 07:00:01 PM
12799191661.986	DD Mmm YY	16 May 88
	DD MMM YY HH:MM	16 MAY 88 19:01
	WWW, MMM DD, YYYY ZH:MI AP	MON, MAY 16, 1988 7:01 PM
	Wwwwwwwwwz, ZM/ZD/YY HH:MI:SS.99	Monday, 5/16/88 19:01:01.98
12799191662.009	YYYY	1988
	YY	88
	Y	8
	MM	05
	ZM	5
	RRRR	V␣␣␣
	MMM	MAY
	Mmm	May
	Mmmmmmmmm	May␣␣␣␣␣
	Mmmmmmmmmz	May
	DD	16
	ZD	16
	DDD	137
	HH	19
	ZH	19
	MI	01
	SS	02
	99	00
	999	009
	AP	PM
	WWW	MON
	Www	Mon
	Wwwwwwwww	Monday␣␣␣␣
	Wwwwwwwwz	Monday

Example

- Convert number of seconds to YYYY/MM/DD HH.MM.SS format:

```
CALL CEEDATM (secs, 'YYYY/MM/DD HH.MI.SS', timestmp,
              fc);
```

Convert Seconds to Integers (CEESECI) API

Syntax

```
void CEESECI (input_seconds, output_year,
              output_month, output_day, output_hours,
              output_minutes, output_seconds,
              output_milliseconds, [fc])

             _FLOAT8     *input_seconds;
             _INT4       *output_year;
             _INT4       *output_month;
             _INT4       *output_day;
             _INT4       *output_hours;
             _INT4       *output_minutes
             _INT4       *output_seconds;
             _INT4       *output_milliseconds;
             _FEEDBACK   *fc;
```

The Convert Seconds to Integers (CEESECI) API converts a number representing the number of seconds since 00:00:00 14 October 1582 to seven separate binary integers representing year, month, day, hour, minute, second, and millisecond. Use CEESECI instead of CEEDATM when the output is needed in numeric format rather than in character format.

Parameters

input_seconds (input)
A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86 401 (24*60*60 + 01). The valid range is 86 400 to 265 621 679 999.999 (23:59:59.999 31 December 9999).

If *input_seconds* is not valid, all output parameters are set to zero.

output_year (output)
A 32-bit binary integer representing year. The range is 1582 through 9999.

output_month (output)
A 32-bit binary integer representing month. The range is 1 through 12.

output_day (output)
A 32-bit binary integer representing day. The range is 1 through 31.

output_hours (output)
A 32-bit binary integer representing hours. The range is 0 through 23.

output_minutes (output)
A 32-bit binary integer representing minutes. The range is 0 through 59.

output_seconds (output)
A 32-bit binary integer representing seconds. The range is 0 through 59.

output_milliseconds (output)
A 32-bit binary integer representing milliseconds. The range is 0 through 999.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE2505 Severity: 30	The value for seconds is out of range

Usage Notes

- The inverse of CEESECI is CEEISEC. The CEEISEC API converts integer year, month, day, hour, minute, second, and millisecond to number of seconds.
- If the input value is a Lilian date instead of seconds, multiply the Lilian date by 86 400 (number of seconds in a day), and pass the new value to CEESECI.
- CEESECI can be used to do date arithmetic that cannot otherwise be done with Lilian dates or number of seconds.

Convert Timestamp to Number of Seconds (CEESECS) API

Syntax

```
void CEESECS (input_timestamp, picture_string,
              output_seconds, [fc])

             _CHAR       *input_timestamp;
             _CHAR       *picture_string;
             _FLOAT8     *output_seconds;
             _FEEDBACK   *fc;
```

The Convert Timestamp to Number of Seconds (CEESECS) API converts a string representing a timestamp into a number representing the number of seconds since 00:00:00 14 October 1582. This API makes it easier to do time calculations, such as the elapsed time between two timestamps.

Parameters

input_timestamp (input by descriptor)
A character string representing a date or timestamp in the format shown by *picture_string*. The field width is 5 through 255 characters. *Input-timestamp* may contain leading or trailing blanks. After a valid date or

timestamp is parsed, remaining characters are ignored. Valid dates are in the range 15 October 1582 to 31 December 9999. A full date must be specified. Valid times are in the range 00:00:00.000 to 23:59:59.999.

If any part or all of the time value is omitted, zeros are substituted for the remaining values. For example,

```
1988-05-17-19:02 is equivalent to 1988-05-17-19:02:00
1988-05-17       is equivalent to 1988-05-17-00:00:00
```

picture_string (input by descriptor)

A character string indicating the format of the date or timestamp value in *input_char_date*, for example `YY/MM/DD HH.MI.SS`. Each character in *picture_string* represents a character in *input_char_date*. If delimiters such as the slash (/) appear in the picture string, leading zeros may be omitted. For example, these calls assign the same value to the variable `secs`.

```
CALL CEESECS('88/06/03 15.35.03',
             'YY/MM/DD HH.MI.SS', secs, fc);

CALL CEESECS('88/6/3 15.35.03'  ,
             'YY/MM/DD HH.MI.SS', secs, fc);

CALL CEESECS('88/6/3 3.35.03 PM',
             'YY/MM/DD HH.MI.SS AP', secs, fc);

CALL CEESECS('88.155 3.35.03 pm',
             'YY.DDD HH.MI.SS AP', secs, fc);
```

See Figure 35-1 on page 35-3 for a list of valid picture characters, and Figure 35-2 on page 35-4 for examples of valid picture strings.

If *picture_string* is null or blank, CEESECS obtains *picture_string* based upon the current job value for COUNTRY ID. For example, if the current job value for COUNTRY ID is US (United States), the date format would be `MM/DD/YYYY`. If the current job value for COUNTRY ID is FR (France), the date format is `DD.MM.YYYY`.

This default mechanism makes it easy not only for translators to specify the preferred date format, but also for application programs and library routines to automatically use this format.

output_seconds (output)

A 64-bit double floating-point number representing the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second 86 401 (24*60*60 + 01). 19:00:01.12 on 16 May 1988 is second 12 799 191 601.12. The largest value that can be represented is 23:59:59.999 on 31 December 9999, which is second 265 621 679 999.999.

Note: A 64-bit double floating-point value can accurately represent approximately 16 significant decimal digits without loss of precision. Therefore, accuracy is available to the nearest millisecond (15 decimal digits).

If *input_timestamp* does not contain a valid date or timestamp, *output_seconds* is set to 0 and CEESECS ends with a nonzero feedback code.

fc (output/optional)

An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling routine. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0501 Severity: 30	The operational descriptor data type is not valid
CEE0502 Severity: 30	Missing operational descriptor
CEE2508 Severity: 30	The value for day is not valid
CEE2509 Severity: 30	The value for era is not valid
CEE2510 Severity: 30	The value for hour is not valid
CEE2513 Severity: 30	The value for Lilian date is not valid
CEE2516 Severity: 30	The value for minute is not valid
CEE2517 Severity: 30	The value for month is not valid
CEE2518 Severity: 30	The picture string specification is not valid
CEE2519 Severity: 30	The value for second is not valid
CEE2521 Severity: 30	The value for year is not valid
CEE2525 Severity: 30	Timestamp picture mismatch

Usage Notes

- The inverse of CEESECS is CEEDATM. The CEEDATM API converts *output_seconds* to character format.
- Elapsed time calculations can be performed easily on the *output_seconds*, because it represents elapsed time. Leap year and end-of-year anomalies are avoided.
- See "Set Century (CEESCEN) API" on page 35-16 and "Query Century (CEEQCEN) API" on page 35-14 for information on 2-digit years.
- If *picture_string* includes a Japanese era symbol `<JJJ>`, the `YY` position in *input_timestamp* is assumed to contain the year within Japanese era. See Figure 35-2 on page 35-4 for an example. See Figure 35-3 on page 35-5 for a list of the Japanese eras recognized by CEESECS.
- If *picture_string* includes an ROC era symbol `<CCC>` or `<CCCCCCCC>`, the `YY` position in *input_timestamp* is

assumed to contain the year within ROC era. See Figure 35-2 on page 35-4 for an example. See Figure 35-4 on page 35-5 for a list of the ROC eras recognized by CEESECS.

Example

- Calculate the difference between two timestamps, in hours:

```
CALL CEESECS ('19880516190001','YYYYMMDDHHMISS',
              secs1, fc);
CALL CEESECS ('1988-05-17-03.00.01',
              'YYYY-MM-DD-HH.MI.SS', secs2, fc);
              diff = (secs2 - secs1) / 3600;
              /* Assume floating-point division */
```

- Convert a timestamp to number of seconds, to calculate the date and time 36 hours ago:

```
now = '1988/07/26 19:55:00';
CALL CEESECS (now, 'YYYY/MM/DD HH:MI:SS', secs, fc);
secs = secs - 36*60*60;
CALL CEEDATM (secs, 'YYYY/MM/DD HH:MI:SS', before,
              fc);
```

Get Current Greenwich Mean Time (CEEGMT) API

The Get Current Greenwich Mean Time (CEEGMT) API is an alias of CEEUTC. See "Get Universal Time Coordinated (CEEUTC) API" on page 35-13 for details.

Get Current Local Time (CEELOCT) API

Syntax

```
void CEELOCT (output_Lilian, output_seconds,
              output_Gregorian, [fc])

            _INT4       *output_Lilian;
            _FLOAT8     *output_seconds;
            _CHAR23     *output_Gregorian;
            _FEEDBACK   *fc;
```

The Get Current Local Time (CEELOCT) API returns the current local time in three formats: Lilian date (the number of days since 14 October 1582), Lilian timestamp (the number of seconds since 00:00:00 14 October 1582), and Gregorian character string (in the form YYYYMMDDHHMISS999'). These values are compatible with the other ILE date and time APIs and with existing language intrinsic functions. CEELOCT performs the same service, faster, than calling CEEUTC, CEEUTCO, and CEEDATM in succession.

Parameters

output_Lilian (output)
A 32-bit binary integer representing the current *local* date in the Lilian format. That is, day 1 is 15 October 1582, day 148 887 is 4 June 1990. If local time is not available from the system, *output_Lilian* is set to 0 and CEELOCT ends with a nonzero feedback code.

output_seconds (output)
A 64-bit double floating point number representing the current *local* date and time as the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86 401 (24*60*60 + 01). 19:00:01.078 on 4 June 1990 is second number 12 863 905 201.078. If local time is not available from the system, *output_seconds* is set to 0 and CEELOCT ends with a nonzero feedback code.

output_Gregorian (output)
A 17-byte character string in the form YYYYMMDDHHMISS999 representing local year, month, day, hour, minute, second, and millisecond.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE2502 Severity: 30	Local time not available

Usage Notes

- Use CEEUTC to determine Universal Time Coordinated (UTC).
- Use CEEUTCO to obtain the offset from UTC to local time.
- The character value returned by CEELOCT is designed to match that produced by existing language intrinsic functions. The numeric values returned can be used to simplify date calculations.
- If the format of *output_Gregorian* is inappropriate, CEEDATM can be used to convert *output_seconds* to any required format.

Example

- Extract current local date and time in the form YYYYMMDDHHMISS999:

```
CALL CEELOCT (days, secs, localdatetime, fc);
```

Get Offset from Universal Time Coordinated to Local Time (CEEUTCO) API

Syntax

```
void CEEUTCO (offset_hours, offset_minutes,
              offset_seconds, [fc])

            _INT4       *offset_hours;
            _INT4       *offset_minutes;
            _FLOAT8     *offset_seconds;
            _FEEDBACK   *fc;
```

The Get Offset from Universal Time Coordinated to Local Time (CEEUTCO) API provides three values representing the current offset from Universal Time Coordinated (UTC) to local system time. *Offset_seconds* can be used with CEEUTC to calculate local date and time. *Offset_hours* and *offset_minutes* express the offset from UTC in terms of hours and minutes.

Parameters

offset_hours (output)
A 32-bit binary integer representing the offset from UTC to local time, in hours; for Pacific Standard Time *offset_hours* is –8. The range for *offset_hours* is –12 to +13, where +13 is Daylight Savings Time in the +12 time zone. If local time offset is not available, *offset_hours* is set to 0 and CEEUTCO ends with a nonzero feedback code.

offset_minutes (output)
A 32-bit binary integer representing the number of additional minutes that local time is ahead of, or behind, UTC. The range for *offset_minutes* is 0 to 59. If the local time offset is not available, *offset_minutes* is set to 0 and CEEUTCO ends with a nonzero feedback code.

offset_seconds (output)
A 64-bit double floating point (output) number representing the offset from UTC to local time, in seconds. For example, Pacific Standard Time is eight hours behind UTC. If the system is in the Pacific time zone during standard time, CEEUTCO returns –28800 (–8 * 60 * 60). The range for *offset_seconds* is –43200 to +46800. If the local time offset is not available from the system, *offset_seconds* is set to 0 and CEEUTCO ends with a nonzero feedback code.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 The API completed successfully
Severity: 00

CEE2503 UTC offset not available
Severity: 30

Usage Notes

- The values returned by CEEUTCO and CEEUTC can be used together to calculate the local date and time.
- The CEEDATM API can be used to convert number of seconds to character timestamp.

Example

- Extract current UTC and convert to local date and time in YYYY-MM-DD HH.MM.SS format:

```
CALL CEEUTC (days, secs, fc);
CALL CEEUTCO (hrs, mins, secoffset, fc);
secs = secs + secoffset;
CALL CEEDATM (secs, 'YYYY-MM-DD HH.MI.SS', timestmp,
              fc);
```

Get Universal Time Coordinated (CEEUTC) API

Syntax

```
void CEEUTC (output_UTC_Lilian, output_UTC_seconds,
             [fc])

            _INT4       *output_UTC_Lilian;
            _FLOAT8     *output_UTC_seconds;
            _FEEDBACK   *fc;
```

The Get Universal Time Coordinated (CEEUTC) API returns the current Universal Time Coordinated as both a Lilian date and as the number of seconds since 00:00:00 14 October 1582. These values are compatible with the other ILE date and time APIs.

Parameters

output_UTC_Lilian (output)
A 32-bit binary integer representing the current date in Greenwich, England, in the Lilian format. That is, the number of days since 14 October 1582, also known as Universal Time Coordinated (UTC). For example, 16 May 1988 is day number 148138. If UTC is not available from the system, *output_UTC_Lilian* is set to 0 and CEEUTC ends with a nonzero feedback code.

output_UTC_seconds (output)
A 64-bit double floating-point number representing the current date and time in Greenwich, England as the number of seconds since 00:00:00 on 14 October 1582. For example, 00:00:01 on 15 October 1582 is second number 86401 (24*60*60 + 01). 19:00:01.078 on 16 May 1988 is second number 12799191601.078. If UTC is not available from the system, *output_UTC_seconds* is

set to 0 and CEEUTC ends with a nonzero feedback code.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE2531 Severity: 30	UTC not available

Usage Notes

- Use CEELOCT to obtain local time.
- Use CEEUTCO to obtain the offset from UTC to local time.
- The values returned by CEEUTC are handy for "wall-clock" calculations.
- CEEDATE will convert *output_UTC_Lilian* to character date, and CEEDATM will convert *output_UTC_seconds* to character timestamp.

Example

- Extract current UTC and convert to local date and time in YYMMDDHHMMSS format:

```
CALL CEEUTC (days, secs, fc);
CALL CEEUTCO (hrs, mins, secoffset, fc);
secs = secs + secoffset;
CALL CEEDATM (secs, 'YYMMDDHHMISS', localdatetime,
              fc);
```

Query Century (CEEQCEN) API

Syntax

```
void CEEQCEN (century_start, [fc])

          _INT        *century_start;
          _FEEDBACK   *fc;
```

The Query Century (CEEQCEN) API queries the century within which 2-digit year values are assumed to lie. Use it in conjunction with CEESCEN when it is necessary to save and restore the current setting. If the ILE default is in effect, all 2-digit years are assumed to lie within a 100-year window. This window starts 80 years prior to the system date and CEEQCEN returns a value of 80.

Parameters

century_start (output)
An integer between 0 and 100. A value of 80 (the ILE default) means all 2-digit years lie within the 100-year window starting 80 years before the system date. For example, in 1990, all 2-digit years are assumed to represent dates between 1910 and 2009, inclusive.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully

Return Default Date and Time Strings for Country (CEEFMDT) API

Syntax

```
void CEEFMDT ([country_code], datetime_str, [fc])

          _CHAR2      *country_code;
          _CHAR       *datetime_str;
          _FEEDBACK   *fc;
```

The Return Default Date and Time String for Country (CEEFMDT) API returns the default date and time picture strings for the country specified in the country_code parameter.

Parameters

country_code (input/optional)
The 2-character string that represents the country code. Figure 35-7 on page 35-15 contains the country codes. If this value is blank, the default country code is used.

datetime_str (output by descriptor)
The default date and time picture string for the country code is placed into this character string variable.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully

CEE3470 Severity: 20	The resulting string is truncated
CEE3471 Severity: 30	The country_code identifier is not valid for CEEFMDT

Country code identifiers

The following table lists the country code identifiers.

Figure 35-7. Country Codes

Country	ID
Albania	AL
Algeria	DZ
Argentina	AR
Australia	AU
Austria	AT
Bahrain	BH
Belgium	BE
Bolivia	BO
Brazil	BR
Bulgaria	BG
Canada	CA
Chile	CL
Colombia	CO
Costa Rica	CR
Countries of the former Yugoslavia	YU
Czech Republic	CZ
Denmark	DK
Dominican Republic	DO
Ecuador	EC
Egypt	EG
El Salvador	SV
Finland	FI
France	FR
Germany	DE
Greece	GR
Guatemala	GT
Honduras	HN
Hungary	HU
Iceland	IS
India	IN
Iran	IR
Iraq	IQ
Ireland	IE
Israel	IL
Italy	IT
Japan	JP
Jordan	JO
Korea, Democratic People's Republic	KP
Korea, Republic of	KR
Kuwait	KW
Lebanon	LB
Libya	LY
Mexico	MX
Morocco	MA
Netherlands	NL
New Zealand	NZ
Norway	NO
Oman	OM
Pakistan	PK
Panama	PA
Paraguay	PY
People's Republic of China	CN
Peru	PE
Poland	PL
Portugal	PT
Qatar	QA
Republic of China	TW
Romania	RO
Russia	RU
Saudi Arabia	SA
Slovakia	SK
Slovenia	SI
South Africa	ZA
Spain	ES
Sudan	SD
Sweden	SE
Switzerland	CH
Syria	SY
Thailand	TH
Tunisia	TN
Turkey	TR
United Arab Emirates	AE
United Kingdom	GB
United States	US
Uruguay	UY
Venezuela	VE
Yemen	YE

Return Default Date String for Country (CEEFMDA) API

The Return Default Date String for Country (CEEFMDA) API returns the default date picture string for the country specified in the country_code parameter.

Syntax

```
void CEEFMDA ([country_code], date_pic_str, [fc])

             _CHAR2      *country_code;
             _CHAR       *date_pic_str;
             _FEEDBACK   *fc;
```

Parameters

country_code (input/optional)
The 2-character string that represents the country code. Figure 35-7 on page 35-15 contains the country codes. If this value is blank, the default country code is used.

date_pic_str (output by descriptor)
The default date picture string for the country code is placed into this character string variable.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE3466 Severity: 20	The date picture string is truncated
CEE3467 Severity: 30	The country_code identifier is not valid for CEEFMDA

Return Default Time String for Country (CEEFMTM) API

The Return Default Time String for Country (CEEFMTM) API returns the default time picture string for the country specified in the country_code parameter.

Syntax

```
void CEEFMTM ([country_code], time_pic_str, [fc])

             _CHAR2      *country_code;
             _CHAR       *time_pic_str;
             _FEEDBACK   *fc;
```

Parameters

country_code (input/optional)
The 2-character string that represents the country code. Figure 35-7 on page 35-15 contains the country codes. If this value is blank, the default country code will be used.

time_pic_str (output)
The default time picture string for the country code is placed into this character string variable.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE3468 Severity: 20	The time picture string is truncated
CEE3469 Severity: 30	The country_code identifier is not valid for CEEFMTM

Set Century (CEESCEN) API

The Set Century (CEESCEN) API sets the century within which 2-digit year values are assumed to lie. Use it in conjunction with CEEDAYS or CEESECS when processing date values that contain 2-digit years (in the YYMMDD format), and when the ILE default century interval does not meet the requirements of a particular application.

Syntax

```
void CEESCEN (century_start, [fc])

             _INT4       *century_start;
             _FEEDBACK   *fc;
```

Parameters

century_start (input)
An integer between 0 and 100. A value of 80 (the ILE default) means all 2-digit years lie within the 100-year window starting 80 years before the system date. For example, in 1990, all 2-digit years are assumed to represent dates between 1910 and 2009, inclusive.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, a condition token is returned to the calling procedure. If not specified and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE2502 Severity: 30	Local time not available
CEE2533 Severity: 30	The century_start value is not between 0 - 100

Chapter 36. Math APIs

The ILE math bindable APIs can be called either through the intrinsic functions of an ILE high-level language, or from an ILE HLL as a call to the API.

The math bindable APIs are:

- Absolute Function (CEESxABS)
- Arccosine (CEESxACS)
- Arcsine (CEESxASN)
- Arctangent (CEESxATN)
- Arctangent2 (CEESxAT2)
- Conjugate of Complex (CEESxCJG)
- Cosine (CEESxCOS)
- Cotangent (CEESxCTN)
- Error Function and Its Complement (CEESxERx)
- Exponential Base e (CEESxEXP)
- Exponentiation (CEESxXPx)
- Factorial (CEE4SIFAC)
- Floating Complex Divide (CEESxDVD)
- Gamma Function (CEESxGMA)
- Hyperbolic Arctangent (CEESxATH)
- Hyperbolic Cosine (CEESxCSH)
- Hyperbolic Sine (CEESxSNH)
- Hyperbolic Tangent (CEESxTNH)
- Imaginary Part of Complex (CEESxIMG)
- Log Gamma Function (CEESxLGM)
- Logarithm Base 10 (CEESxLG1)
- Logarithm Base 2 (CEESxLG2)
- Logarithm Base e (CEESxLOG)
- Modular Arithmetic (CEESxMOD)
- Nearest Integer (CEESxNIN)
- Nearest Whole Number (CEESxNWN)
- Positive Difference (CEESxDIM)
- Sine (CEESxSIN)
- Square Root (CEESxSQT)
- Tangent (CEESxTAN)
- Transfer of Sign (CEESxSGN)
- Truncation (CEESxINT)
- Basic Random Number Generation (CEERAN0)

Except for the Basic Random Number Generation (CEERAN0) API, the APIs are presented in alphabetical order.

The messages that can be generated by the math bindable APIs are summarized in "Message Descriptions" on page 36-12.

Data Types and Limits

Following is a description of the data types used in the math bindable APIs. The special values and limits of the data types are also listed.

The semantics and requirements of a high-level language may affect the values of the data types.

Integer Data Types

Figure 36-1 shows the range of values that math bindable APIs can represent and use with the integer data types.

Figure 36-1. Precise Limit Values of Integer Data Types

Data Type	Length (bytes)	Range of values
_INT2	2	-32 768 through 32 767
_INT4	4	-2 147 483 648 through 2 147 483 647

The operating system represents integers internally in two's complement notation, and the leftmost bit is the sign of the number.

Real Data Types

Figure 36-2 shows the range of values that math bindable APIs can represent and use with the real data types.

Figure 36-2. Approximate Limit Values of Real Data Types

Data Type	Length (bytes)	Approximate Absolute Nonzero Minimum	Approximate Absolute Maximum	Approximate Precision (Decimal Digits)
_FLOAT4	4	1.175494×10^{-38}	3.402823×10^{38}	7
_FLOAT8	8	$2.225074 \times 10^{-308}$	1.797693×10^{308}	15

The operating system represents real data type values in IEEE floating-point format.

Figure 36-3 lists the special values for floating-point operations. The values adhere to the IEEE standard for binary floating-point arithmetic.

Figure 36-3 (Page 1 of 2). Floating-point Special Values

Data Type	Description	Values
_FLOAT4	+INF	0x7F800000
_FLOAT4	HUGE	0x7F7FFFFF
_FLOAT4	+0	0x00000000
_FLOAT4	-0	0x80000000
_FLOAT4	-INF	0xFF800000

Figure 36-3 (Page 2 of 2). Floating-point Special Values

Data Type	Description	Values
_FLOAT4	masked NaN	0x7FC00000
_FLOAT4	unmasked NaN	0x7F800010
_FLOAT8	+INF	0x7FF00000 0x00000000
_FLOAT8	HUGE	0x7FEFFFFF 0xFFFFFFFF
_FLOAT8	+0	0x00000000 0x00000000
_FLOAT8	-0	0x80000000 0x00000000
_FLOAT8	-INF	0xFFF00000 0x00000000
_FLOAT8	masked NaN	0x7FF80000 0x00000000
_FLOAT8	unmasked NaN	0x7FF00000 0x00000001

Complex Data Types

Figure 36-4 shows the range of values that the math bindable APIs can represent and can use with complex data types.

Figure 36-4. Approximate Limit Values of Complex Data Types

Data Type	Length (bytes)	Approximate Values
_COMPLEX8	8	Both real and imaginary parts are _FLOAT4. See Figure 36-2 on page 36-1
_COMPLEX16	16	Both real and imaginary parts are _FLOAT8. See Figure 36-2 on page 36-1

Syntax Conventions for Math Bindable APIs

The x within the API name represents a data type. The possible values for x are:

- **I** A 32-bit binary integer. The parameter value is _INT4.
- **S** A 32-bit single floating-point number. The parameter value is _FLOAT4.
- **D** A 64-bit double floating-point number. The parameter value is _FLOAT8.
- **T** A 32-bit single floating-complex number (both real and imaginary parts are 32 bits long). The parameter value is _COMPLEX8.
- **E** A 64-bit double floating-complex number (both real and imaginary parts are 64 bits long). The parameter value is _COMPLEX16.

The fc parameter in the syntax for the math APIs represents an optional 12-byte feedback code.

Math Bindable API Procedures

The math bindable APIs are procedures, and therefore parameters are passed by reference. The math library handles all exceptions by returning a feedback code, if one was specified on the call. If the pointer to the feedback area is null (0), the feedback area is created and signaled to the ILE condition manager and a program message is generated by the condition manager.

The APIs that perform computations will do so in the rounding mode of "round to nearest" to achieve better accuracy. The APIs save the current rounding mode (before the computations start), and restore the previous one before the APIs exit. This is a general rule for the math bindable APIs.

The following notation is used in the individual APIs:

- **|x|** denotes the absolute value of x.
- **sign(x)** is +1 if $x \geq 0$ or -1 if $x < 0$.
- **f** denotes a function result.
- **z** denotes a complex argument, where $z = x + iy$.
- **z1 and z2** denote two complex arguments, where $z1 = x1 + iy1$ and $z2 = x2 + iy2$.

Absolute Function (CEESxABS) API

The Absolute Function (CEESxABS) API returns the absolute value of the parameter using:

$f = |x| = (x > 0) \ ? \ x : -x$

or

$f = |z| = u$

where

$u = x \times \sqrt{1.0 + (\frac{y}{x})^2}$ if $x > y$;

$u = y \times \sqrt{1.0 + (\frac{x}{y})^2}$ if $x \leq y$.

Syntax–Integer Data Types

```
void CEESIABS(_INT4 *x, _INT4 *result, _FEEDBACK *fc);
```

Syntax–Real Data Types

```
void CEESSABS(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDABS(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTABS(_COMPLEX8 *z, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESEABS(_COMPLEX16 *z, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Integer Data Types

Any integer number.

Real Data Types

Any real number.

Complex Data Types

Any complex number.

Arccosine (CEESxACS) API

The Arccosine (CEESxACS) API returns the arccosine of the parameter, using:

$f = \cos^{-1}(x)$

Syntax–Real Data Types

```
void CEESSACS(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDACS(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

$|x| \leq 1$

Arcsine (CEESxASN) API

The Arcsine (CEESxASN) API returns the arcsine of the parameter, using:

$f = \sin^{-1}(x)$

Syntax–Real Data Types

```
void CEESSASN(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDASN(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

$|x| \leq 1$

Arctangent (CEESxATN) API

The Arctangent (CEESxATN) API returns the arctangent of the parameter, using:

$f = \tan^{-1}(x)$

or

$f = \tan^{-1}(z) = \tan^{-1}(x+iy) = u + iv$

where

$u = \frac{1}{2}\,\text{atan2}(2x, 1 - x^2 - y^2)$

$v = \frac{1}{4}\ln\frac{(1+y)^2 + x^2}{(1-y)^2 + x^2}$

Syntax–Real Data Types

```
void CEESSATN(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDATN(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTATN(_COMPLEX8 *x, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESEATN(_COMPLEX16 *x, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument.

Complex Data Types

Any complex argument.

Arctangent2 (CEESxAT2) API

The Arctangent2 (CEESxAT2) API calculates the arctangent of the first parameter divided by the second parameter, using:

$f = \text{atan2}(\frac{y}{x})$

For $x > 0$, atan2(y, x) = atan(y/x).

For $x < 0$ and $y > 0$, atan2(y, x) = π + atan(y/x).

For x < 0 and y < 0, atan2(y, x) = -π + atan(x/y).

atan2(+0,-1) = atan2(1,-INF) = π
atan2(-0,-1) = atan2(-1,-INF) = -π
atan2(+0,1) = atan2(1,+INF) = +0
atan2(-0,1) = atan2(-1,+INF) = -0

atan2(1,0) = atan2(+INF,±1) = π/2

atan2(-1,0) = atan2(-INF,±1) = -π/2

Syntax–Real Data Types

```
void CEESSAT2(_FLOAT4 *y, _FLOAT4 *x,
              _FLOAT4 *result, _FEEDBACK *fc);
void CEESDAT2(_FLOAT8 *y, _FLOAT8 *x,
              _FLOAT8 *result, _FEEDBACK *fc);
```

Argument Range

Real Data Types

y ≠ 0.0 and x ≠ 0.0; y ≠ ±INF and x ≠ ±INF

Conjugate of Complex (CEESxCJG) API

The Conjugate of Complex (CEESxCJG) API returns the conjugate of a complex number, using:

f = x - *i*y for argument z = x + *i*y.

Syntax–Complex Data Types

```
void CEESTCJG(_COMPLEX8 *x, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESECJG(_COMPLEX16 *x, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Complex Data Types

Any complex number.

Cosine (CEESxCOS) API

The Cosine (CEESxCOS) API returns the cosine of the parameter, using:

f = cos(x)

or

f = cos(z) = cos(x+*i*y)
= (cos(x)*cosh(y)) - *i*(sin(x)*sinh(y))

Syntax–Real Data Types

```
void CEESSCOS(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDCOS(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTCOS(_COMPLEX8 *x, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESECOS(_COMPLEX16 *x, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument such that |x| ≤ 0x432921FB54442D18 ≈ π × 2^{50} ≈ 3537118876014220.0 .

Complex Data Types

Any complex number such that |y| ≤ 88.7228 for _FLOAT4 and |y| ≤ 709.7827 for _FLOAT8, and x is any real argument.

Cotangent (CEESxCTN) API

The Cotangent (CEESxCTN) API returns the cotangent of the parameter, using:

f = cot(x)

Syntax–Real Data Types

```
void CEESSCTN(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDCTN(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument such that |x| ≤ 0x432921FB54442D18 ≈ π × 2^{50} ≈ 3537118876014220.0 .

Error Function and Its Complement (CEESxERx) API

The Error Function and Its Complement (CEESxERx) API calculates the error function and its complement, using:

$$f = erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t} dt$$

Syntax–Real Data Types

```
void CEESSERF(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDERF(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
void CEESSERC(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDERC(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument.

Exponential Base e (CEESxEXP) API

The Exponential Base e (CEESxEXP) API performs the mathematical function of e raised to a power, using:

$$f = e^x$$

or

$$f = e^z = e^{x+iy}$$
$$= (e^x * \cos(y)) + i(e^x * \sin(y))$$

Syntax–Real Data Types

```
void CEESSEXP(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDEXP(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTEXP(_COMPLEX8 *z, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESEEXP(_COMPLEX16 *z, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

-87.3365 ≤ x ≤ 88.7228 for single precision;
-708.3964 ≤ x ≤ 709.7827 for double precision

Complex Data Types

-87.3365 ≤ x ≤ 88.7228 and y is any real number for single precision;
-708.3964 ≤ x ≤ 709.7827 and y is any real number for double precision: $|y| \le 0x432921FB54442D18 \approx \pi \times 2^{50} \approx 3537118876014220.0$

Exponentiation (CEESxXPx) API

The Exponentiation (CEESxXPx) API performs the mathematical function exponentiation, using:

$$f = x^y$$

where x and y have the following combinations:

x	y
integer	integer
real _FLOAT4	integer
real _FLOAT8	integer
single complex	integer
double complex	integer
real _FLOAT4	real _FLOAT4
real _FLOAT8	real _FLOAT8
single complex	single complex
double complex	double complex

Syntax–Integer Data Types

```
void CEESIXPI(_INT4 *x, _INT4 *y, _INT4 *result,
              _FEEDBACK *fc);
void CEESSXPI(_FLOAT4 *x, _INT4 *y, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDXPI(_FLOAT8 *x, _INT4 *y, _FLOAT8 *result,
              _FEEDBACK *fc);
void CEESTXPI(_COMPLEX8 *z1, _INT4 *x,
              _COMPLEX8 *result, _FEEDBACK *fc);
void CEESEXPI(_COMPLEX16 *z1, _INT4 *x,
              _COMPLEX16 *result, _FEEDBACK *fc);
```

Syntax–Real Data Types

```
void CEESSXPS(_FLOAT4 *x, _FLOAT4 *y,
              _FLOAT4 *result, _FEEDBACK *fc);
void CEESDXPD(_FLOAT8 *x, _FLOAT8 *y,
              _FLOAT8 *result, _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTXPT(_COMPLEX8 *z1, _COMPLEX8 *z2,
              _COMPLEX8 *result, _FEEDBACK *fc);
void CEESEXPE(_COMPLEX16 *z1, _COMPLEX16 *z2,
              _COMPLEX16 *result, _FEEDBACK *fc);
```

Argument Range

Integer Data Types

Any integer arguments.

Real Data Types

Any real argument subject to the condition that if x is negative, y must be an integer.

Complex Data Types

See the argument for real data types above.

Factorial (CEE4SIFAC) API

The Factorial (CEE4SIFAC) API returns the result of a factorial of an integer, using:

$n! = 1*2*3 \ldots (n-1)*n.$

Syntax–Integer Data Types

```
void CEE4SIFAC(_INT4 *x, _INT4 *result,
               _FEEDBACK *fc);
```

Argument Range

Integer Data Types

Any positive integer ≤ 30.

Floating Complex Divide (CEESxDVD) API

The Floating Complex Divide (CEESxDVD) API returns the result of the division of two complex numbers, using:

$f = z_1 \div z_2 = (x_1 + iy_1) \div (x_2 + iy_2)$
$= u + iv$

where
if $|x_2| \geq |y_2|$

$$u = \frac{x_1 + \frac{y_2}{x_2} \times y_1}{x_2 + \frac{y_2}{x_2} \times y_2}$$

$$v = \frac{y_1 - \frac{y_2}{x_2} \times x_1}{x_2 + \frac{y_2}{x_2} \times y_2}$$

if $|x_2| < |y_2|$

$$u = \frac{x_1 \times \frac{x_2}{y_2} + y_1}{y_2 + \frac{x_2}{y_2} \times x_2}$$

$$v = \frac{y_1 \times \frac{x_2}{y_2} - x_1}{y_2 + \frac{x_2}{y_2} \times x_2}$$

Syntax–Complex Data Types

```
void CEESTDVD(_COMPLEX8 *x, _COMPLEX8 *y,
              _COMPLEX8 *result, _FEEDBACK *fc);
void CEESEDVD(_COMPLEX16 *x, _COMPLEX16 *y,
              _COMPLEX16 *result, _FEEDBACK *fc);
```

Argument Range

Complex Data Types

Any complex number.

Floating Complex Multiply (CEESxMLT) API

The Floating Complex Multiply (CEESxMLT) API returns the product of two complex numbers, using:

$f = z_1 \times z_2 = (x_1 + iy_1) \times (x_2 + iy_2)$
$= u + iv$

where
$u = x_1 \times x_2 - y_1 \times y_2$

$v = y_1 \times x_2 + x_1 \times y_2$

Syntax–Complex Data Types

```
void CEESTMLT(_COMPLEX8 *x, _COMPLEX8 *y,
              _COMPLEX8 *result, _FEEDBACK *fc);
void CEESEMLT(_COMPLEX16 *x, _COMPLEX16 *y,
              _COMPLEX16 *result, _FEEDBACK *fc);
```

Argument Range

Complex Data Types

Any complex number.

Gamma Function (CEESxGMA) API

The Gamma Function (CEESxGMA) API uses the mathematical gamma function, using:

$$f = \Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt$$

Syntax–Real Data Types

```
void CEESSGMA(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDGMA(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

$0 < x \leq 35.04$ for _FLOAT4 and $0 < x \leq 171.6243$ for _FLOAT8.

Hyperbolic Arctangent (CEESxATH) API

The Hyperbolic Arctangent (CEESxATH) API returns the hyperbolic arctangent of the parameter, using:

$$f = \tanh^{-1}(x) = \frac{1}{2} \times \ln \frac{(1+x)}{(1-x)}$$

or

$$f = \tanh^{-1}(z) = \tanh^{-1}(x+iy) = u + iv$$

where

$$u = \frac{1}{4} \times \ln \frac{(1+x)^2 + y^2}{(1-x)^2 + y^2}$$

$$v = \frac{1}{2} \times \text{atan2}(2y,\ 1 - x^2 - y^2)$$

Syntax–Real Data Types

```
void CEESSATH(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDATH(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTATH(_COMPLEX8 *z, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESEATH(_COMPLEX16 *z, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument such that $|x| \leq 1.0$.

Complex Data Types

Any complex number such that $x \neq 1.0$ and $y \neq 0$.

Hyperbolic Cosine (CEESxCSH) API

The Hyperbolic Cosine (CEESxCSH) API returns the hyperbolic cosine of the parameter, using:

$$f = \cosh(x) = \frac{(e^x + e^{-x})}{2}$$

or

$$f = \cosh(z) = \cosh(x+iy)$$
$$= \cosh(x)*\cos(y) + i\sinh(x)*\sin(y)$$

Syntax–Real Data Types

```
void CEESSCSH(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDCSH(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTCSH(_COMPLEX8 *z, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESECSH(_COMPLEX16 *z, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument such that $|x| \leq 89.4159$ for _FLOAT4 and 709.7827 for _FLOAT8.

Complex Data Types

Any complex number such that $|x| \leq 89.4159$ for _FLOAT4 and 709.7827 for _FLOAT8.

Hyperbolic Sine (CEESxSNH) API

The Hyperbolic Sine (CEESxSNH) API performs the mathematical function hyperbolic sine, using:

$$f = \sinh(x) = \frac{(e^x - e^{-x})}{2}$$

or

$$f = \sinh(z) = \sinh(x+iy)$$
$$= \sinh(x) \times \cos(y) + i\cosh(x) \times \sin(y)$$

Syntax–Real Data Types

```
void CEESSSNH(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDSNH(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTSNH(_COMPLEX8 *z, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESESNH(_COMPLEX16 *z, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument such that $|x| \leq 89.4159$ for _FLOAT4 and 709.7827 for _FLOAT8.

Complex Data Types

Any complex number such that $|x| \leq 89.4159$ for _FLOAT4 and 709.7827 for _FLOAT8.

Hyperbolic Tangent (CEESxTNH) API

The Hyperbolic Tangent (CEESxTNH) API performs the function hyperbolic tangent, using:

$$f = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

or

$$f = \tanh(z) = \tanh(x+iy)$$
$$= \frac{\sinh(2x) + i\sin(2y)}{\cosh(2x) + \cos(2y)}$$

Syntax–Real Data Types

```
void CEESSTNH(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDTNH(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTTNH(_COMPLEX8 *z, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESETNH(_COMPLEX16 *z, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument

Complex Data Types

Any complex number such that $|x| < 43.66825$ for single precision, and
$|x| < 354.1982$ for double precision.

Imaginary Part of Complex (CEESxIMG) API

The Imaginary Part of Complex (CEESxIMG) API returns the imaginary part of a complex number, using:

$f = y$, where $z = x + iy$.

Syntax–Complex Data Types

```
void CEESTIMG(_COMPLEX8 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESEIMG(_COMPLEX16 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Complex Data Types

Any complex number.

Log Gamma Function (CEESxLGM) API

The Log Gamma Function (CEESxLGM) API performs the mathematical function of Log Gamma, using:

$$f = \ln(\Gamma(x))$$

Syntax–Real Data Types

```
void CEESSLGM(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDLGM(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

$0 < x \leq 4.0850 \times 10^{36}$ for _FLOAT4 and,
$0.0 < x \leq 2^{1014}$ for _FLOAT8.

Logarithm Base 10 (CEESxLG1) API

The Logarithm Base 10 (CEESxLG1) API performs the mathematical function of logarithm base 10, using:

$f = \log_{10}(x) = \ln(x) * \log_{10}(e)$

Syntax–Real Data Types

```
void CEESSLG1(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDLG1(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

$x > 0.0$

Logarithm Base 2 (CEESxLG2) API

The Logarithm Base 2 (CEESxLG2) API performs the mathematical function of logarithm base 2, using:

$$f = \log_2(x) = \ln(x) * \frac{1}{\ln(2)}$$

Syntax–Real Data Types

```
void CEESSLG2(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDLG2(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

$x > 0.0$

Logarithm Base e (CEESxLOG) API

The Logarithm Base e (CEESxLOG) API performs the mathematical function of logarithm base e, using:

$f = \log_e(x)$ or $\ln(x)$

or

$f = \ln(z) = \log_e(|z|) + i\text{atan2}(y, x)$

Syntax–Real Data Types

```
void CEESSLOG(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDLOG(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTLOG(_COMPLEX8 *z, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESELOG(_COMPLEX16 *z, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

$x > 0.0$

Complex Data Types

$z \neq 0 + i0$.

Modular Arithmetic (CEESxMOD) API

The Modular Arithmetic (CEESxMOD) API performs the mathematical function modular arithmetic, using:

f = remainder of x / y

The absolute value of the result is always less than the absolute value of y.

Syntax–Integer Data Types

```
void CEESIMOD(_INT4 *x, _INT4 *y, _INT4 *result,
              _FEEDBACK *fc);
```

Syntax–Real Data Types

```
void CEESSMOD(_FLOAT4 *x, _FLOAT4 *y,
              _FLOAT4 *result, _FEEDBACK *fc);
void CEESDMOD(_FLOAT8 *x, _FLOAT8 *y,
              _FLOAT8 *result, _FEEDBACK *fc);
```

Argument Range

Integer Data Types

Any two integer numbers such that y ≠ 0.

Real Data Types

Any two real numbers such that y ≠ 0.0.

Nearest Integer (CEESxNIN) API

The Nearest Integer (CEESxNIN) API performs the mathematical function nearest integer, using:

If x ≥ 0.0
  f = sign(x)*n where n is the largest integer ≤ |x + 0.5|.

If x < 0.0
  f = sign(x)*n where n is the largest integer ≤ |x - 0.5|.

Syntax–Real Data Types

```
void CEESSNIN(_FLOAT4 *x, _INT4 *result,
              _FEEDBACK *fc);
void CEESDNIN(_FLOAT8 *x, _INT4 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real number.

Nearest Whole Number (CEESxNWN) API

The Nearest Whole Number (CEESxNWN) API performs the mathematical function nearest whole number, using:

If x ≥ 0.0
  f = sign(x)*n where n is the largest integer ≤ |x + 0.5|.

If x < 0.0
  f = sign(x)*n where n is the largest integer ≤ |x - 0.5|.

Syntax–Real Data Types

```
void CEESSNWN(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDNWN(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real number.

Positive Difference (CEESxDIM) API

The Positive Different (CEESxDIM) API returns the positive difference between two numbers, using:

If x > y, f = x - y

or

If x ≤ y, f = 0

Syntax–Integer Data Types

```
void CEESIDIM(_INT4 *x, _INT4 *y, _INT4 *result,
              _FEEDBACK *fc);
```

Syntax–Real Data Types

```
void CEESSDIM(_FLOAT4 *x, _FLOAT4 *y,
              _FLOAT4 *result, _FEEDBACK *fc);
void CEESDDIM(_FLOAT8 *x, _FLOAT8 *y,
              _FLOAT8 *result, _FEEDBACK *fc);
```

Argument Range

Integer Data Types

Any integer argument.

Real Data Types

Any real number.

Sine (CEESxSIN) API

The Sine (CEESxSIN) API returns the sine of the parameter, using:

f = sin(x)

or

f = sin(z) = sin(x+*i*y)
  = (sin(x)*cosh(y)) + *i*(cos(x)*sinh(y))

Syntax–Real Data Types

```
void CEESSSIN(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDSIN(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTSIN(_COMPLEX8 *x, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESESIN(_COMPLEX16 *x, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument such that $|x| \leq 0x432921FB54442D18 \approx \pi \times 2^{50} \approx 3537118876014220.0$.

Complex Data Types

Any complex number such that $|y| \leq 88.7228$ for _FLOAT4, and $|y| \leq 709.7827$ for _FLOAT8, and $|x| \leq 0x432921FB54442D18 \approx \pi \times 2^{50} \approx 3537118876014220.0$.

Square Root (CEESxSQT) API

The Square Root (CEESxSQT) API returns the square root of the parameter, using:

$f = \sqrt{x}$

or

$f = \sqrt{z} = \sqrt{x + iy} = u + iv$

Syntax–Real Data Types

```
void CEESSSQT(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDSQT(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTSQT(_COMPLEX8 *z, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESESQT(_COMPLEX16 *z, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument such that $x \geq 0.0$.

Complex Data Types

Any complex number such that $|z| + |x| \leq 1.797693*10^{308}$.

Tangent (CEESxTAN) API

The Tangent (CEESxTAN) API returns the tangent of the parameter, using:

$f = \tan(x)$

or

$f = \tan(z) = \tan(x+iy)$

$$= \frac{\sin(2x) + i\sinh(2y)}{\cos(2x) + \cosh(2y)}$$

Syntax–Real Data Types

```
void CEESSTAN(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDTAN(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Syntax–Complex Data Types

```
void CEESTTAN(_COMPLEX8 *z, _COMPLEX8 *result,
              _FEEDBACK *fc);
void CEESETAN(_COMPLEX16 *z, _COMPLEX16 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real argument such that $|x| \leq 0x432921FB54442D18 \approx \pi \times 2^{50} \approx 3537118876014220.0$.

Complex Data Types

Any complex number such that $|y| < 43.66825$ for single precision, and $|y| < 354.1982$ for double precision.

Transfer of Sign (CEESxSGN) API

The Transfer of Sign (CEESxSGN) API performs the mathematical function transfer of sign, using:

$f = \text{sign}(y)|x|$

Syntax–Integer Data Types

```
void CEESISGN(_INT4 *x, _INT4 *y, _INT4 *result,
              _FEEDBACK *fc);
```

Syntax–Real Data Types

```
void CEESSSGN(_FLOAT4 *x, _FLOAT4 *y,
              _FLOAT4 *result, _FEEDBACK *fc);
void CEESDSGN(_FLOAT8 *x, _FLOAT8 *y,
              _FLOAT8 *result, _FEEDBACK *fc);
```

Argument Range

Integer Data Types

Any integer argument.

Real Data Types

Any real number.

Truncation (CEESxINT) API

The Truncation (CEESxINT) API returns the truncated value of the parameter, using:

f = sign(x) * n, where n is the largest integer ≤ |x|.

Syntax–Real Data Types

```
void CEESSINT(_FLOAT4 *x, _FLOAT4 *result,
              _FEEDBACK *fc);
void CEESDINT(_FLOAT8 *x, _FLOAT8 *result,
              _FEEDBACK *fc);
```

Argument Range

Real Data Types

Any real number.

Message Descriptions

A summary of the messages generated by the math bindable APIs is contained in Figure 36-5.

Figure 36-5 (Page 1 of 2). Messages Generated by Math Bindable APIs

Msg_No	Msg_Id	Explanation
2002	CEE2002	The argument is too close to multiple (pi/2)'s for tangent and multiple pi's for cotangent. This causes an inaccurate result.
2003	CEE2003	In (x**y) both x and y are integers, x = 0 and y < 0 for procedure &1. The result is undefined. The fixed-point zero-divide exception occurs.
2004	CEE2004	In (x**y) x is real and y is integer, x = 0.0 and y < 0 for procedure &1. The result is undefined. The floating-point zero-divide exception occurs.
2005	CEE2005	The value of the argument for &1 is outside the range &2. It causes a floating-point overflow exception.
2006	CEE2006	In (x**y) both x and y are real, x = 0.0 and y < 0.0 for procedure &1. The result is undefined. The floating-point zero-divide exception occurs.
2008	CEE2008	For an exponentiation operation (Z**P) where the complex base Z equals 0, the real part of the complex exponent P is less than or equal to 0. The floating-point zero-divide exception occurs in procedure &1.
2009	CEE2009	The value of the real part of a complex argument is greater than &2 for procedure &1. It causes a floating-point overflow exception.
2010	CEE2010	The value of the argument is less than 0.0 for procedure &1. It is not valid for the square root function. The result is undefined.
2011	CEE2011	The argument for procedure &1 is greater than &2. It causes a floating-point overflow exception.
2012	CEE2012	The argument for procedure &1 is negative. It is not valid for the logarithmic function.
2013	CEE2013	The absolute value of the imaginary part of the complex argument for &1 is greater than &2. This causes floating-point overflow.
2014	CEE2014	Both arguments to the arctangent2 function are either 0 or infinity for procedure &1. They are not valid for arctangent2 function.
2015	CEE2015	The value of the real part of a complex argument for procedure &1 is less than &2. It causes a floating-point underflow exception.
2016	CEE2016	The absolute value of the argument for procedure &1 is greater than &2. The argument is out of range and the result is undefined.

Figure 36-5 (Page 2 of 2). Messages Generated by Math Bindable APIs

Msg_No	Msg_Id	Explanation
2017	CEE2017	The absolute value of the argument for procedure &1 is greater than or equal to &2. This causes a floating-point overflow exception.
2018	CEE2018	The real and imaginary parts of the argument for procedure &1 are zero or infinity. They are not valid for complex logarithmic functions.
2019	CEE2019	The absolute value of the real part of the complex argument for procedure &1 is greater than &2. This causes a floating-point overflow exception.
2020	CEE2020	In (x**y) x < 0.0 and y is not an integer for procedure &1. It causes a floating-point zero-divide exception. The result is undefined.
2022	CEE2022	The complex argument is not valid for procedure &1. It may be one of the following: The real part of the argument is 0. Real**2 + imaginary**2 = 1.0.
2023	CEE2023	The calculated result overflows the result field in procedure &1.
2024	CEE2024	Floating-point overflow exception occurred in procedure &1.
2025	CEE2025	Floating-point underflow exception occurred in procedure &1.
2026	CEE2026	The denominator is 0. The operand for the modular function is not valid. It causes a floating-point zero-divide exception.
2027	CEE2027	Floating-point zero divide exception occurred in procedure &1.
2101	CEE2101	The argument is an unmasked NaN for procedure &1. If the argument is a complex number, either its real part or imaginary part is an unmasked NaN. It causes a floating-point incorrect operand exception.
2102	CEE2102	The argument for procedure &1 is less than &2. It causes a floating-point underflow exception.
2103	CEE2103	Floating-point operand exception occurred in &1.
2117	CEE2117	The values of the real part and imaginary part cannot be 1.0 and 0.0 respectively at the same time for procedure &1. The result is undefined.
2118	CEE2118	The sum of the absolute value of the complex number, and the absolute value of its real part, is greater than the maximum FLOAT8 (1.797693*10**308) for procedure &1.

Additional Math API

Basic Random Number Generation (CEERAN0) API

The Basic Random Number Generation (CEERAN0) API generates a sequence of uniform pseudorandom numbers between 0 and 1 using the multiplicative congruential method with a user-specified seed.

Syntax

```
void CEERAN0 (seed, random_no, [fc])

            _INT4       *seed;
            _FLOAT8     *random_no;
            _FEEDBACK   *fc;
```

seed (input/output)
A 32-bit binary integer representing an initial value used to generate random numbers. It must be a variable; that is, it cannot be an input-only parameter. The valid range is 0 to 2147483646.

If the value of the seed parameter is 0, the seed is generated from the current Greenwich Mean Time.

On return, CEERAN0 changes the value of *seed* so that it may be used as the new seed in the next call.

random_no (output)
A 64-bit double floating-point pseudorandom number with a value between 0 and 1, exclusive. If *seed* is not valid, *random_no* is set to –1 and CEERAN0 ends with a nonzero feedback code.

fc (output/optional)
An optional 12-byte feedback code passed by reference. If specified as an argument, feedback information (a condition token) is returned to the calling procedure. If not specified, and the requested operation was not successfully completed, the condition is signaled to the condition manager.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE2523 Severity: 10	UTC not available to generate random seed from system time
CEE2524 Severity: 30	Seed value for &1 is not valid

Chapter 37. Message Services APIs

Bindable APIs are provided for message services. The APIs can be used to dispatch, format, obtain, retrieve, and store messages.

The message services APIs are:

Dispatch a Message (CEEMOUT) dispatches a message string.
Get a Message (CEEMGET) retrieves a message and stores it in a buffer.
Get, Format, and Dispatch a Message (CEEMSG) obtains, formats, and dispatches a message that corresponds to an input condition token.

The APIs in this chapter are presented in alphabetical order.

Dispatch a Message (CEEMOUT) API

Syntax

```
void CEEMOUT (Message_String, Destination_Code, [fc])

            _VSTRING    *Message_String;
            _INT4       *Destination_Code;
            _FEEDBACK   *fc;
```

The Dispatch a Message (CEEMOUT) API is used to dispatch a message string.

Parameters

Message_String (input by descriptor)
A valid ILE string, passed by reference with a descriptor, to be dispatched as a message. It is not necessary that this string first be retrieved by the CEEMGET API. Insert data cannot be inserted with this call.

Destination_Code (input)
A 4-byte binary integer of one of the following values:

1 Dispatch the message for output to the standard output (console or file). The message is also logged in the *EXT message queue of the job.

2 Log the message to the system message file only; the message is not displayed.

Note: Messages sent through this API are subject to the normal AS/400 rules for message filtering.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0451 Severity: 30	Unsupported destination code &2 passed to procedure &1
CEE0501 Severity: 30	The operational descriptor data type is not valid
CEE0502 Severity: 30	Missing operational descriptor
CEE3101 Severity: 30	&1 cannot be called in the default activation group
CEE9902 Severity: 30	Unexpected user error occurred in &1

Get a Message (CEEMGET) API

Syntax

```
void CEEMGET (Cond_Token, Message_Area, Msg_Ptr,
              [fc])

            _FEEDBACK   *Cond_Token;
            _VSTRING    *Message_Area;
            _INT4       *Msg_Ptr;
            _FEEDBACK   *fc;
```

The Get a Message (CEEMGET) API retrieves a message and stores it in a buffer for manipulation or output by the caller.

The API retrieves a message and places it in the storage location referenced by the Message_Area parameter.

The Msg_Ptr parameter has a value of zero on the initial call to the CEEMGET API. If the message is too large to be contained in *Message_Area*, *Msg_Ptr* is returned containing an index into the message. The index is used in subsequent calls to CEEMGET to retrieve the remaining portion of the message. When the entire message has been retrieved, *Msg_Ptr* is returned containing a value of zero.

Parameters

Cond_Token (input)
A 12-byte condition token. See "How Conditions Are Represented" on page 34-1 for a description of the condition token.

Message_Area (output by descriptor)
A valid ILE string variable, passed by reference with a descriptor. The CEEMGET API places the retrieved message into this string variable.

Msg_Ptr (input/output)
A 4-byte integer with a value of 0 on the initial call to CEEMGET to retrieve a message. If the message is too large to be contained in the *Message_Area*, *Msg_Ptr* will be returned containing an index into the message. The index is used in subsequent calls to CEEMGET to

retrieve the remaining portion of the message. When the entire message has been retrieved, *Msg_Ptr* is returned with a value of 0.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0102 Severity: 30	The condition token passed to &1 is not valid
CEE0454 Severity: 30	&1 cannot find message &3 in message file &2
CEE0455 Severity: 10	The message returned is truncated
CEE0458 Severity: 30	&1 cannot find message file &2
CEE0501 Severity: 30	The operational descriptor data type is not valid
CEE0502 Severity: 30	Missing operational descriptor
CEE3103 Severity: 30	Cannot allocate storage in &1
CEE9902 Severity: 30	Unexpected user error occurred in &1

Usage Notes

- If *Msg_Ptr* is greater than the length of the message being retrieved, then *Msg_Ptr* is set to 0, and a zero-length string is copied into *Message_Area*.
- Insert data cannot be inserted with this call.

Get, Format, and Dispatch a Message (CEEMSG) API

Syntax

```
void CEEMSG (Cond_Token, Destination_Code, [fc])

            _FEEDBACK   *Cond_Token;
            _INT4       *Destination_Code;
            _FEEDBACK   *fc;
```

The Get, Format, and Dispatch a Message (CEEMSG) API is used to obtain, format, and dispatch a message corresponding to an input condition token.

Parameters

Cond_Token (input)
A 12-byte condition token. See "How Conditions Are Represented" on page 34-1 for a description of the condition token.

Destination_Code (input)
A 4-byte binary integer of one of the following values:

1 Dispatch the message for output to the standard output. The message is also logged in the system message file.

2 Log the message to the system message file only; the message is not displayed.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0102 Severity: 30	The condition token passed to &1 is not valid
CEE0451 Severity: 30	Unsupported destination code &2 passed to procedure &1
CEE0458 Severity: 30	&1 cannot find message file &2
CEE9902 Severity: 30	Unexpected user error occurred in &1

Chapter 38. Program or Procedure Call APIs

Bindable APIs are provided to retrieve operational descriptor information and to test for omitted arguments.

The program or procedure call APIs are:

Get String Information (CEEGSI) retrieves string information about a parameter.
Retrieve Operational Descriptor Information (CEEDOD) retrieves operational descriptor information about a parameter.
Test for Omitted Argument (CEETSTA) tests for the presence or absence of an omittable argument.

The APIs are presented in alphabetical order.

Get String Information (CEEGSI) API

The Get String Information (CEEGSI) API retrieves string information about a parameter used in the call to this API. String information describes the elements of a parameter, such as the type and the length of the string.

Syntax

```
void CEEGSI (posn, datatype, currlen, maxlen, [fc])

             _INT4       *posn;
             _INT4       *datatype;
             _INT4       *currlen;
             _INT4       *maxlen;
             _FEEDBACK   *fc;
```

Parameters

posn (input)
The ordinal position in the parameter list of the formal parameter whose operational descriptor is required. A value of 1 denotes the leftmost parameter.

datatype (output)
The binary value of the data type field. The possible values and their data types are:

1 typeEsc
An element descriptor type (descElmt) that is not one of the following data types.
2 typeChar
A string of SBCS characters with values in the range X'00' through X'FF'.
3 typeCharZ
A string of SBCS characters with values in the range X'01' through X'FF' that ends with a null byte (X'00').
4 typeCharV2
A string of SBCS characters prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.
5 typeCharV4
A string of SBCS characters prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.
6 typeBit
A string of bits with values of 0 or 1.
7 typeBitV2
A string of bits prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of bits.
8 typeBitV4
A string of bits prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of bits.
9 typeGChar
A string of DBCS characters with values in the range X'0000' through X'FFFF'.
10 typeGCharZ
A string of DBCS characters with values in the range X'0001' through X'FFFF' that end with a null byte (X'0000').
11 typeGCharV2
A string of DBCS characters prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.
12 typeGCharV4
A string of DBCS characters prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.

currlen (output)
A binary number containing the current number of elements in the string, as follows:

- For strings of typeEsc, this is the length from the descriptor, and is equal to the maximum length returned in *maxlen*.
- For strings of typeCharV2 and typeCharV4, this is the length from the argument (binary prefix) itself.
- For strings of typeCharZ, the API determines the current length (in number of characters, either SBCS or DBCS) of the string by scanning for the null character. If the length in the descriptor is nonzero (the compiler of the caller knew the extent), the search is confined to that length. Otherwise, the scan continues until a null character is found.

- If the data type was undefined, the contents of this field are undefined.

maxlen (output)
A binary number containing the maximum number of string elements, as follows:

- For strings of typeEsc, this is the length from the descriptor, and is equal to the current length returned in *currlen*.
- For strings of typeCharV2 and typeCharV4, this is the length from the descriptor (which does not include the length of the prefix).
- For strings of typeCharZ, the maximum length is the number of the characters excluding the null character. It is the maximum length from the descriptor minus 1 (to account for the SBCS or DBCS null character). If the length in the descriptor is zero, the maximum length is set equal to the current length.
- If the data type was undefined, the contents of this field are undefined.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0501 Severity: 30	The operational descriptor data type is not valid
CEE0502 Severity: 30	Missing operational descriptor
CEE0505 Severity: 10	A NULL-terminated string is not found

Usage Notes

- It is an error to use CEEGSI to test an argument that is not a reference argument or that is preceded in the argument list by other arguments that are not reference arguments.

  Note: This error may not be diagnosed.
- The results are undefined if this API is used in a function that does not have operational descriptors specified.
- The address of this API cannot be taken.

Retrieve Operational Descriptor Information (CEEDOD) API

Syntax

```
void CEEDOD (posn, desctype, datatype, descinf1,
             descinf2, datalen, [fc])

             _INT4       *posn;
             _INT4       *desctype;
             _INT4       *datatype;
             _INT4       *descinf1;
             _INT4       *descinf2;
             _INT4       *datalen;
             _FEEDBACK   *fc;
```

The Retrieve Operational Descriptor Information (CEEDOD) API retrieves operational descriptor information about a parameter used in the call to this API. The operational descriptor communicates additional information about a parameter, such as size and shape.

Parameters

posn (input)
The ordinal position in the parameter list of the formal parameter whose operational descriptor is required. A value of 1 denotes the leftmost parameter.

desctype (output)
The binary value of the descriptor type field. The possible values and their descriptor types are:

1 descEsc
An escape descriptor.
2 descElmt
An element descriptor. Elements are objects such as numbers and strings, that can be aggregated into arrays and structures.
3 descArray
An array descriptor.
4 descStruct
A structure descriptor.

datatype (output)
The binary value of the data type field. The possible values and their data types are:

1 typeEsc
An element descriptor type (descElmt) that is not one of the following data types.
2 typeChar
A string of SBCS characters with values in the range X'00' through X'FF'.
3 typeCharZ
A string of SBCS characters with values in the range X'01' through X'FF' that ends with a null byte (X'00').
4 typeCharV2
A string of SBCS characters prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.

5 typeCharV4
A string of SBCS characters prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.

6 typeBit
A string of bits with values of 0 or 1.

7 typeBitV2
A string of bits prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of bits.

8 typeBitV4
A string of bits prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of bits.

9 typeGChar
A string of DBCS characters with values in the range X'0000' through X'FFFF'.

10 typeGCharZ
A string of DBCS characters with values in the range X'0001' through X'FFFF' that end with a null byte (X'0000').

11 typeGCharV2
A string of DBCS characters prefixed by an unsigned 2-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.

12 typeGCharV4
A string of DBCS characters prefixed by an unsigned 4-byte binary count field. The count field specifies the current length of the string in terms of the number of string elements, that is, the number of characters.

descinf1 (output)
The binary value of the first descriptor information field. If the descriptor omits this field, *descinf1* is set to 0.

descinf2 (output)
The binary value of the second descriptor information field (used for bit alignment, for example). If the descriptor omits this field, *descinf2* is set to 0.

datalen (output)
The 4-byte binary value of the descriptor length field.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0501 Severity: 30	The operational descriptor data type is not valid
CEE0502 Severity: 30	Missing operational descriptor

Usage Notes

- The address of this API cannot be taken.
- It is an error to use CEEDOD to test an argument that is not a reference argument. It is also an error to test an argument that is preceded in the argument list by other arguments that are not reference arguments.
- The results are undefined if you use the CEEDOD API in a function that does not specify operational descriptors.

Test for Omitted Argument (CEETSTA) API

The Test for Omitted Argument (CEETSTA) API is used to test for the presence or absence of an omittable argument.

Syntax

```
void CEETSTA (presence_flag, arg_num, [fc])

             _INT4       *presence_flag;
             _INT4       *arg_num;
             _FEEDBACK   *fc;
```

Parameters

presence_flag (output)
The presence or absence of the argument tested. A value of 1 indicates that the argument is present; a value of 0 indicates that the argument was omitted.

arg_num (input)
The argument number to be tested (in the argument list of the procedure that called the procedure where the CEETSTA call is coded), counting the first argument as 1.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0503 Severity: 30	The argument number is not valid **Note:** ILE does not check the condition where the argument number is greater than the number of arguments in the list.
CEE3005 Severity: 30	The argument being tested is not an address

Usage Notes

- It is an error to use CEETSTA to test an argument that is not a reference argument or that is preceded in the argument list by other arguments that are not reference arguments.

  Note: This error may not be diagnosed.

- The address of this API cannot be taken.

Chapter 39. Storage Management APIs

Bindable APIs are provided for all heap operations. Applications can be written using either the storage management APIs, language-intrinsic functions, or both.

The storage management APIs fall into the following categories:

- Basic heap operations

  The Free Storage (CEEFRST) API frees one previously allocated heap storage.

  The Get Heap Storage (CEEGTST) API allocates storage within a heap.

  The Reallocate Storage (CEECZST) API changes the size of previously allocated storage.

- Extended heap operations

  The Create Heap (CEECRHP) API creates a new heap.

  The Discard Heap (CEEDSHP) API discards an existing heap.

  The Mark Heap (CEEMKHP) API returns a token that can be used to identify heap storage to be freed by the CEERLHP API.

  The Release Heap Storage (CEERLHP) API frees all storage allocated in the heap since the mark was specified.

- Heap allocation strategies

  The Define Heap Allocation Strategy (CEE4DAS) API defines an allocation strategy that determines the attributes for a heap created with the CEECRHP API.

Allocation Strategy Type (_CEE4ALC)

The _CEE4ALC allocation strategy type contains attributes that are used to define the characteristics of the storage allocated for heaps. The attributes are shown in Figure 39-1.

User-Defined Allocation Strategy

You can define an allocation strategy by altering the attributes of the _CEE4ALC allocation strategy type. You do this with the Define Heap Allocation Strategy (CEE4DAS) API. Then, when you use the Create Heap (CEECRHP) API, you can specify the allocation strategy that you defined for the heap attributes you require.

If you use the CEECRHP API, but did not define an allocation strategy, a default allocation strategy in terms of _CEE4ALC provides the heap attributes.

Note: The creation size and extension size values of _CEE4ALC may be overridden. You do this by specifying values for the initial_size parameter and the increment parameter on the CEECRHP bindable API.

The _CEE4ALC allocation strategy type contains the following attributes.

```
struct _CEE4ALC {
          _INT4 Max_Sngl_Alloc;  /* maximum size of a single allocation */
          _INT4 Min_Bdy;         /* minimum boundary alignment of any allocation */
          _INT4 Crt_Size;        /* initial creation size of the heap */
          _INT4 Ext_Size;        /* the extension size of the heap */
          _INT2 reserved1;       /* must be binary 0 */
          _BITS Alloc_Strat:1;   /* a choice for allocation strategy */
          _BITS No_Mark:1;       /* a group deallocation choice */
          _BITS Blk_Xfer:1;      /* a choice for block transfer of a heap */
          _BITS PAG:1;           /* a choice for heap creation in a PAG */
          _BITS Alloc_Init:1;    /* a choice for allocation initialization */
          _CHAR Init_Value;      /* initialization value */
          _BITS reserved2:3;     /* must be binary 0 */
          _BITS reserved3:32;    /* must be binary 0 */
         };
```

Figure 39-1. _CEE4ALC Definition

Max_Sngl_Alloc
The maximum allocation size, in bytes, of any single allocation from the heap. This attribute is useful for controlling the use of the heap. The minimum value for this attribute is 4 bytes, and the maximum value is 16MB minus 64KB. If 0 is specified, the default value of 16MB minus 64KB is used.

Min_Bdy
The minimum boundary alignment, in bytes, associated with any allocation from the heap. The minimum value for a boundary alignment is 4 bytes, and maximum value is 512 bytes. To allow valid pointers to be stored in a storage allocation with the heap, a minimum boundary alignment of 16 bytes is required. If zero is specified, a default value of 16 bytes is used for the boundary align-

ment. The minimum boundary alignment is rounded up to a power of 2.

Crt_Size
The creation size, in bytes, of the heap. The minimum value for the size of the heap is 512 bytes and the maximum value is 16MB minus 1KB. If 0 is specified, the system computes a default value. The value is rounded up to a 512-byte boundary.

Note: If system resources are constrained, the system may override the value specified.

Ext_Size
The extension size of the heap in bytes. The minimum value for extension size is 512 bytes and the maximum value is 16MB minus 1KB. If 0 is specified the system computes a default value. The value is rounded up to a 512-byte boundary.

Note: If system resources are constrained, the system may override the value specified.

reserved1
Must be binary 0.

Alloc_Strat
Allows a choice between:

0 Normal allocation strategy.
1 Create a process space on each allocation.

Note: This option should be used only in unusual situations, such as in debugging application problems caused by references past the end of an allocation.

No_Mark
Allows a choice between:

0 Allow the use of the CEEMKHP and CEERLHP APIs.
1 Do not allow the use of the CEEMKHP and CEERLHP APIs.

Blk_Xfer
Used to increase the performance of a heap based on prior knowledge of how the heap is used. This attribute is used only when the heap is not a member of a process access group (PAG). The values are:

0 Transfer the minimum storage transfer size (that is, 1 storage unit).
1 Transfer the machine default storage transfer size (that is, 8 storage units).

PAG
A heap can be created as a process access group (PAG) member. The values are:

0 Do not create the heap in the PAG.
1 Create the heap in the PAG.

Note: It is possible for the PAG to overflow, at which point any requested PAG heap creations or extensions will not reside in the PAG. Therefore, the system may ignore the request to create the heap in the PAG.

Alloc_Init
Allows the user to specify if all storage allocations from the heap being created will be initialized to the initial value. The values are:

0 Do not initialize the heap with Init_Value.
1 Initialize the heap with Init_Value.

Init_Value
The value used to initialize the storage allocations.

reserved2
Must be binary 0.

reserved3
Must be binary 0.

The Default Heap

From the programmer's viewpoint, a default heap is always available in the activation group. In fact, the first request to allocate storage results in the creation of the default heap from which the storage allocation takes place.

The attributes of the default heap are defined by the system through a default allocation strategy. You cannot change this default allocation strategy. Following is an example of the default allocation strategy:

```
Max_Sngl_Alloc = 16MB - 64KB
Min_Bdy        = 16
Crt_Size       = 4KB
Ext_Size       = 4KB
Alloc_Strat    = 0
No_Mark        = 1
Blk_Xfer       = 0
PAG            = 0
Alloc_Init     = 0
Init_Value     = 0x00  /* This value is not used */
                       /* if Alloc_Init is 0     */
```

In addition, the default heap in an activation group has the following special characteristics:

- It may not be discarded by CEEDSHP (discard a heap); it is guaranteed to be present for the life of the activation group.
- It is referred to with a heap identifier of 0.

Languages that do not have an intrinsic multiple-heap storage model (such as ILE C/400) use the default heap. This heap cannot be discarded and is immune to the Mark Heap (CEEMKHP) and Release Heap (CEERLHP) APIs. Storage allocated within the default heap can be freed only by explicit free operations or when the owning activation group ends.

This implementation ensures that the storage is not inadvertently released in mixed-language applications. Release heap and discard heap operations are considered insecure for the following reasons:

- Large applications that re-use existing code with potentially different storage models.
- The programmer is not completely familiar with the internals of each procedure.

If release heap operations were valid for the default heap, then procedures that correctly use different storage management capabilities separately might fail when used in combination.

Create Heap (CEECRHP) API

Syntax

```
void CEECRHP (heap_id, [initial_size], [increment],
              [alloc_strat_ID], [fc])

             _INT4       *heap_id;
             _INT4       *initial_size;
             _INT4       *increment;
             _INT4       *alloc_strat_ID;
             _FEEDBACK   *fc;
```

The Create Heap (CEECRHP) API creates a new heap.

Parameters

heap_id (output)
The heap identifier of the created heap.

initial_size (input/optional)
The initial amount of storage, in bytes, allocated for the new heap. If this parameter is 0 or omitted, and there is no corresponding value in the user-defined allocation strategy, the system computes a default value. The minimum value that can be specified is 512 bytes. If values between 1 and 512 bytes are specified, the system rounds the value up to 512 bytes automatically. The maximum value that can be specified is 16MB minus 1KB. The value specified is rounded up to a 512-byte boundary.

Note: If system resources are constrained, the system may override the value specified.

increment (input/optional)
The number of bytes by which the heap is extended when it is necessary to enlarge the heap to satisfy an allocation request. If this parameter is 0 or omitted, and there is no corresponding value in the user-defined allocation strategy, the system computes a default value. The minimum value that can be specified is 512 bytes. If values between 1 and 512 bytes are specified, the system rounds the value up to 512 bytes automatically. The maximum value that can be specified is 16MB minus 1KB. The value specified is rounded up to a 512-byte boundary.

Note: If system resources are constrained, the system may override the value specified.

alloc_strat_ID (input/optional)
The allocation strategy used. ILE allows allocation strategy values of 0, 1, and 40 through 44 from the possible range of values 0 through 99, where:

- **0** Specifies use of an allocation strategy that is the optimal default for the system.
- **1** Same as 0.
- **40-44** User-defined allocation strategies. See "Define Heap Allocation Strategy (CEE4DAS) API" on page 39-4 for information on defining allocation strategies.

For allocation strategy values outside of the values specified above:

- **2-39** The values are reserved for other systems. If they are specified, message CEE0814 is issued.
- **45-49** The values are reserved for other systems. If they are specified, message CEE0815 is issued.
- **50-99** The values are not supported by ILE. If they are specified, message CEE0806 is issued.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

Severities of 10 or less represent success. For higher severities, no heap is created, and all other output values are undefined.

CEE0000 Severity: 00	The API completed successfully
CEE0804 Severity: 30	The initial_size parameter is not valid
CEE0805 Severity: 30	The increment parameter is not valid
CEE0806 Severity: 30	Allocation strategy identifier value is not valid
CEE0809 Severity: 30	The maximum number of heaps supported by ILE has been reached
CEE0813 Severity: 30	Insufficient storage available to satisfy the request
CEE0814 Severity: 30	Allocation strategy identifier is not supported by ILE
CEE0815 Severity: 30	Allocation strategy identifier is not supported by ILE
CEE3006 Severity: 30	At least one field in the allocation strategy is not valid

Usage Notes

- If a user-defined allocation strategy value is specified for which the CEE4DAS API was not called, the default user-created heap strategy is used. See "User-Defined Allocation Strategy" on page 39-1.
- The *initial_size* and *increment* values, if specified, will override any default or user-created heap strategy values.
- If one of the fields in the allocation strategy CEE4DAS is not valid, the heap is not created and an error condition is raised. This API checks the allocation strategy.

Define Heap Allocation Strategy (CEE4DAS) API

Syntax

```
void CEE4DAS (alloc_strat_ID, alloc_strat_in,
              [alloc_strat_out], [fc])

             _INT4       *alloc_strat_ID;
             _CEE4ALC    *alloc_strat_in;
             _CEE4ALC    *alloc_strat_out;
             _FEEDBACK   *fc;
```

The Define Heap Allocation Strategy (CEE4DAS) API defines a system-specific allocation strategy and associates the defined strategy with a specified allocation strategy identifier. When creating a heap using the CEECRHP API, the allocation strategy identifier can be used within the activation group in which it was defined.

Parameters

alloc_strat_ID (input)
The allocation strategy identifier being defined. The valid values are 40 through 44.

alloc_strat_in (input)
A structure of allocation strategy type _CEE4ALC that defines the new allocation strategy to be assigned to the specified allocation strategy identifier. See "Allocation Strategy Type (_CEE4ALC)" on page 39-1 for a description.

alloc_strat_out (output/optional)
An optional structure of type _CEE4ALC that will be set to the value of the previous allocation strategy that was assigned to the specified allocation strategy identifier. If no previous allocation strategy was assigned, the default allocation strategy is returned. See "User-Defined Allocation Strategy" on page 39-1 for more information.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0816 Severity: 30	The allocation strategy identifier is out of range

Usage Notes

- The CEE4DAS API does not perform a validity check on the contents of *alloc_strat_in*. When the defined allocation strategy is used, checking is enforced by the CEECRHP API.

Discard Heap (CEEDSHP) API

Syntax

```
void CEEDSHP (heap_id, [fc])

             _INT4       *heap_id;
             _FEEDBACK   *fc;
```

The Discard Heap (CEEDSHP) API deletes an existing heap.

Parameters

heap_id (input)
The heap identifier of the heap to be discarded.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0803 Severity: 30	The heap identifier does not match any existing heap
CEE0812 Severity: 30	The basic initial heap cannot be marked and discarded

Usage Notes

- A *heap_id* of 0 is not valid. This is the heap identifier of the default heap; it cannot be discarded.
- After this call, there may still be pointers to storage that had been allocated from this heap. Their use can cause unpredictable or erroneous results.

Free Storage (CEEFRST) API

Syntax

```
void CEEFRST (address, [fc])

             _POINTER    *address;
             _FEEDBACK   *fc;
```

The Free Storage (CEEFRST) API frees previously allocated heap storage.

Parameters

address (input)
The address returned by a previous CEEGTST call or a language-intrinsic function. The storage at this address is deallocated.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0802 Severity: 40	The storage headers are damaged
CEE0810 Severity: 30	The starting address for reallocation is not valid

Usage Notes

- The heap identifier is inferred from the address of the storage to be freed. The storage is deallocated from the heap in which it was allocated. The deallocate operation may be issued from an activation group other than the activation group that owns the heap.
- The pointer to the address passed as the argument is no longer valid after this call. The storage freed by this operation can be reallocated on a subsequent CEEGTST call. If the pointer is not reassigned, any further use of it will have unpredictable results.

Get Heap Storage (CEEGTST) API

Syntax

```
void CEEGTST (heap_id, size, address, [fc])

              _INT4       *heap_id;
              _INT4       *size;
              _POINTER    *address;
              _FEEDBACK   *fc;
```

The Get Heap Storage (CEEGTST) API allocates storage within a heap.

Parameters

heap_id (input)
A heap identifier of the heap in which the storage is allocated.

size (input)
The number of bytes of storage to be allocated.

address (output)
The address of the first byte of allocated storage.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0802 Severity: 40	The storage headers are damaged
CEE0803 Severity: 30	The heap identifier does not match any existing heap
CEE0808 Severity: 30	Requested storage size was not valid
CEE0813 Severity: 30	Insufficient storage available to satisfy the request

Usage Notes

- The size value is rounded up to a multiple of the minimum boundary specified when the heap space is created. The minimum boundary for the activation group default heap is 16 bytes.
- The address of the first byte of the allocation begins on a boundary at least as great as the minimum boundary specified when the heap is created. The minimum boundary for the activation group default heap is 16 bytes.
- A *heap_id* of 0 specifies the default heap in the activation group.
- Individual allocations within a heap may not be contiguous.
- Each allocation associated with a heap provides a sequence of contiguously addressable bytes. The bytes do not cross a 64KB boundary unless the allocation size is greater than 64KB.

Mark Heap (CEEMKHP) API

Syntax

```
void CEEMKHP (heap_id, mark, [fc])

              _INT4       *heap_id;
              _POINTER    *mark;
              _FEEDBACK   *fc;
```

The Mark Heap (CEEMKHP) API returns a token, which can be used with the Release Heap (CEERLHP) API to free a heap storage group. After a CEEMKHP call, all storage subsequently obtained by the CEEGTST API can be freed by a single CEERLHP call. A heap may have multiple marks.

Parameters

heap_id (input)
A heap identifier token specifying the heap to be marked.

mark (output)
The mark token for use in a subsequent CEERLHP call.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0803 Severity: 30	The heap identifier does not match any existing heap
CEE0812 Severity: 30	The basic initial heap cannot be marked and discarded

Usage Notes

- The default heap in the activation group cannot be marked.
- Multiple marks are maintained for each heap.

  When a release operation is performed, the specified mark and all subsequent marks are removed from the list of active marks.
- A mark should only be used in release operations. It should not be used as a pointer.

Reallocate Storage (CEECZST) API

Syntax

```
void CEECZST (address, new_size, [fc])

            _POINTER    *address;
            _INT4       *new_size;
            _FEEDBACK   *fc;
```

The Reallocate Storage (CEECZST) API changes the size of a previously allocated storage block, preserving the contents.

Parameters

address (input/output)
On input, this parameter contains an address returned by a previous CEEGTST call or a language intrinsic function. On output, the address of the first byte of the newly allocated storage is returned in this parameter.

In effect, reallocation of a storage block is accomplished by allocating a new storage block, of size *new_size*, and copying the contents of the old block to the new block.

new_size (input)
The number of bytes of storage to be allocated for the new storage block. This value is rounded up to a multiple of the minimum boundary specified when the heap was created. The minimum boundary for the activation group default heap is 16 bytes.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

A message severity of 10 or less represents success. If the severity is greater than 10:

- No storage is allocated.
- The previous allocation remains intact.
- The value in *address* remains unchanged.

CEE0000 Severity: 00	The API completed successfully
CEE0802 Severity: 40	The storage headers are damaged
CEE0808 Severity: 30	Requested storage size was not valid
CEE0810 Severity: 30	The starting address for reallocation is not valid
CEE0813 Severity: 30	Insufficient storage available to satisfy the request

Usage Notes

- The heap identifier is inferred from the address. The new storage block is allocated from the same heap that contained the old block. The reallocate operation may be issued from an activation group other than the one that owns the heap.
- The contents of the old storage block are preserved in the following way:
  - If *new_size* is greater than the old size, the entire contents of the old storage block are copied to the new block.
  - If *new_size* is less than the old size, the contents of the old block are truncated to the size of the new block.
  - If *new_size* is equal to the old size, the contents of the old storage block are copied to the new block.

  Note: Because the new storage may be allocated at a different location than the existing allocation, any pointers that referred to the old storage are no longer valid. Continued use of such pointers will give unpredictable, or incorrect results.
- Storage that is reallocated maintains the same mark and release status as the old storage block. If the old storage block was marked, the new storage block carries the same mark and is released by a release operation that specifies the mark.

Release Heap (CEERLHP) API

The Release Heap (CEERLHP) API frees all storage allocated since the specified mark in the heap.

Syntax

Syntax

```
void CEERLHP (mark, [fc])

            _POINTER    *mark;
            _FEEDBACK   *fc;
```

Parameters

mark (input)
A mark returned by a previous CEEMKHP call. All storage allocated in the marked heap since the corresponding mark operation is released. Marks obtained after the specified mark are also discarded.

fc (output/optional)
An optional 12-byte feedback code.

Feedback Codes and Conditions

CEE0000 Severity: 00	The API completed successfully
CEE0807 Severity: 30	The mark does not match any existing mark

Usage Notes

- No heap identifier is required for this call. The heap identifier of the heap to be released is inferred from the mark address. The release operation may be issued from an activation group other than the activation group that owns the heap.
- Multiple marks can be maintained for each heap. A release operation deallocates all storage allocated since the specified mark operation.

  Mark and release operations treat the heap in a fashion similar to a stack. There must be a mark operation outstanding at the location used in a release operation. A release operation can reset the heap to any mark; a release operation can clear one or several mark operations.
- Pointers obtained by CEEGTST for storage that is freed by the release operation are no longer valid after this call. Continued use of these pointers will give unpredictable or incorrect results.

Part 11. Message Handling APIs

Chapter 40. 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" on page 40-2. 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*.

The message handling APIs are presented in alphabetical order. "Diagnostic Reporting" on page A-20 shows how to use the APIs to send and receive messages.

The message handling APIs include the following:

Change Exception Message (QMHCHGEM) changes an exception message on a call message queue. This API allows the current program to perform a variety of actions on an exception message that was sent to its caller, a previous caller, or itself.

List Job Log Message (QMHLJOBL) lists messages from a the job message queue of a job. This function gets the requested message information and returns it in a user space in the format specified in the parameter list.

List Nonprogram Messages (QMHLSTM) lists messages from one or two nonprogram message queues. This function gets the requested message information and returns it in a user space in the format specified in the parameter list.

Move Program Messages (QMHMOVPM) moves messages from one call message queue to the message queue of an earlier call stack entry in the call stack. This is especially useful for error handling.

Promote Message (QMHPRMM) promotes an escape or status message that was sent to a call stack entry. That is, the message is handled and replaced with a new escape or status message. You may promote an escape message to another escape message or to a status message. You may promote a status message to an escape message or to another status message.

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 call message queue, and provides 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. This API is similar in function to the Remove Message (RMVMSG) command with the MSGQ parameter.

Remove Program Messages (QMHRMVPM) removes messages from call message queues. 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 call message queue to the message queue of the previous call stack entry in the call 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 Nonprogram Message Queue Attributes (QMHRMQAT) provides information about the attributes of a nonprogram message queue.

Retrieve Request Message (QMHRTVRQ) retrieves request messages from the current job's call 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 call stack entry in the call 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 42, "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 49, "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 call 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 description**, which is stored in a message file (object type *MSGF) on the system. The message description includes the message text and other items, such as message help, the default reply for the message, and its severity level.

Each message description has its own message identifier. The **message identifier** is a 7-character code, such as CPF2469, that refers to the message description 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** is the version of the predefined message that is first displayed at a work station or logged in a job log. The message text is also known as **first-level text**. 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 **call message queues** corresponding to the calls in the job. Each call within a job has a call message queue. If a call stack entry appears in the call stack twice, it has two separate call message queues.

     A call message queue is also known as a program message queue. However, in the ILE model, you can call procedures as well as programs. Both ILE procedure message queues and OPM program message queues are referred to as call message queues.
  3. All message queues other than external and call 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. The message description 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
- 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	NonEPM 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

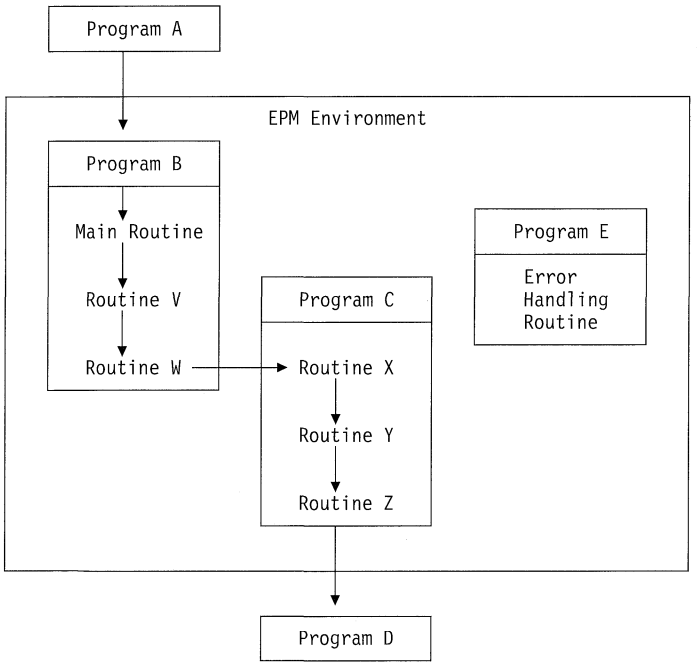
The EPM exception handler can work with the message handling APIs to respond to exceptions that occur in EPM programs running on an AS/400 system. When an error occurs, you can:

- Use the Send Program Message (QMHSNDPM) API to send an escape message from your EPM program's error handling routine to a nonEPM program.
- Use existing EPM functions to resend an escape message from your EPM program's error handling routine to an EPM or nonEPM program.
- Return control from your EPM program's error handling routine to the routine or program that was active before the error occurred. 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 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 nonEPM 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 nonEPM 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" on page 40-5.

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 routine), and all EPM programs are removed from the call 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 call 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 call stack entry in the call 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.

Change Exception Message (QMHCHGEM) API

Parameters

Required Parameter Group:

1	Invocation pointer	Input	Pointer
2	Call stack counter	Input	Binary(4)
3	Message key	Input	Char(4)
4	Modification option	Input	Char(10)
5	Reply text	Input	Char(*)
6	Reply text length	Input	Binary(4)
7	Error code	I/O	Char(*)

The Change Exception Message (QMHCHGEM) API changes an exception message on a call message queue. This API allows the current program to perform one of the following actions on an exception message that was sent to its caller, a previous caller, or itself:

- Handle the escape, status, or notify message. If the exception is a notify message that has not already received a reply, return the default reply.
- If it is a notify message that has not been replied to, return a reply to the call stack entry that sent this message. Handle the exception.
- Handle the escape, status, or notify message and remove it from the job log. If the exception is a notify message that has not been replied to, return its default reply or another reply to the call stack entry that sent it.

Required Parameter Group

Invocation pointer
INPUT; POINTER
The invocation pointer to the call stack entry to which the exception message was sent. When using a value other than 0 for the call stack counter parameter, the invocation pointer points to the call stack entry from which to start counting in the call stack. This will be the location of the call stack entry that received the exception message that was sent. The call stack entry you specify must be in the call stack. A null invocation pointer may be specified. If this pointer is not set, the call stack entry that called the QMHCHGEM API is used.

Call stack counter
INPUT; BINARY(4)
A number that identifies the location in the call stack of the call stack entry that received the message you are changing. The number is relative to the call stack entry specified in the invocation pointer parameter. The number indicates how many calls earlier in the call stack the call stack entry is from the one specified in the invocation pointer parameter.

Valid values follow:

0 Change a message in the message queue of the call stack entry specified in the invocation pointer parameter.

1 Change a message in the message queue of the call stack entry that called the call stack entry specified in the invocation pointer parameter.

n Change a message in the message queue of the nth call stack entry earlier up the stack from the call stack entry specified in the invocation pointer parameter.

You can use any positive number that offsets to an actual call stack entry in the call stack.

Message key
INPUT; CHAR(4)
The message key of the exception message being changed. This parameter is ignored when *CHANGEALL or *CHANGELST is specified for the modification option parameter.

Modification option
INPUT; CHAR(10)
The type of change to be done to the exception message. Valid values follow:

***HANDLE**
Causes the exception to be handled. No error message is returned if the exception is already handled.

If the exception is a status message, it is immediately removed from the job message queue and is no longer accessible. If the exception is an escape or notify message and is still in the job log, it is still accessible through its message key.

If the exception is a notify message, the default reply is sent. If the notify message already received a reply but was not handled, the exception will still be handled. If the notify message was already handled but had not received a reply, the default reply will still be sent.

***CHANGE**
Changes escape message to a diagnostic message and handles it. If the escape message has already been handled, it is still changed to a diagnostic message. If the exception is not an escape message, an error is returned to the caller of this API and the exception is not handled or changed.

***CHANGEALL**
Changes all escape messages that were sent to the specified call stack entry. Each escape message will be changed to a diagnostic message and handled. If the escape message has already been handled, it is still changed to a diagnostic message. If any messages are encountered that are not escape messages, they are ignored. When *CHANGEALL is specified, the message key parameter value is ignored.

***CHANGELST**
Changes the last escape message that was sent to the specified call stack entry. The escape message will be changed to a diagnostic message and handled. If the escape message has already been handled, it is still changed to a diagnostic message. When *CHANGELST is specified, the message key parameter value is ignored.

***REPLY**
Replies to a notify message and handles it. The default reply or the reply specified in the reply text parameter is returned to the call stack entry that sent the notify message. If the reply text length parameter value is 0, the default reply is sent. The notify message is handled after its reply is sent.

If the notify message has already been handled, the reply is still sent.

If the notify message has already been replied to, an error is returned to the caller of this API, and the exception is not handled.

If the exception is not a notify message, an error is returned to the caller of this API, and the exception is not handled.

***REMOVE**
Handles the escape, notify, or status exception and removes the message from the job log. If the exception is a notify message, its default reply or the reply specified in the reply text parameter is returned to the call stack entry that sent it. If the reply text length parameter value is 0, the default reply is sent. No error message is returned if the exception is already handled. The exception is immediately inaccessible for any further operations.

If the exception is a notify message that has already been replied to, it is handled and removed from the job log. No error message is returned to the caller of this API.

If the exception is not a notify message, the reply text length parameter value must be 0. If the reply text length parameter value is not 0, an error is returned to the caller of this API. The exception is not handled or removed from the job log.

Reply text
INPUT; CHAR(*)
The data that is returned as the reply to a notify message when *REPLY or *REMOVE is specified for the modification option parameter. This data must be compatible with the reply type, reply length, and valid reply values stored in the message description for the message. If it is not compatible, an error is returned to the caller of this API.

This parameter is ignored if:

- The message being changed is not a notify message.
- The modification option is not *REPLY or *REMOVE.
- The reply text length parameter is 0.

If 0 is specified for the reply text length parameter, the replay text parameter is ignored.

Reply text length
INPUT; BINARY(4)
The length, in bytes, of the data that is returned as the reply to a notify message. This length must be compatible with the reply type and maximum reply length stored in the message description for the message.

Valid values follow:

0 Return the notify message's default reply.
1–132 Return the reply specified in the reply text parameter. This is the number of bytes to return.

If the message being changed is not a notify message, or the modification option is not *REPLY or *REMOVE, 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

CPF24A3 E Value for call stack counter parameter not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B6 E Length of &1, not valid for message text or data.
CPF2401 E Not authorized to library &1.
CPF2410 E Message key not found in message queue &1.
CPF2411 E Not authorized to message file &1 in &2.
CPF242D E Modification option &1 not valid.
CPF242E E Tried to change message which is not an exception.
CPF242F E Message type must be ESCAPE for *CHANGE modification option.
CPF2420 E Reply already sent for inquiry or notify message.
CPF2422 E Reply not valid.
CPF243A E Invocation pointer parameter not valid.
CPF2432 E Cannot send reply to message type other than *INQ or *NOTIFY.
CPF2547 E Damage to message file QCPFMSG.
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.
CPF9838 E User profile storage limit exceeded.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Job Log Messages (QMHLJOBL) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Message selection information	Input	Char(*)
4	Size of message selection information	Input	Binary(4)
5	Format of message selection information	Input	Char(8)
6	Error code	I/O	Char(*)

The List Job Log Messages (QMHLJOBL) API lists messages sent to the job message queue of a job. This API gets the requested message information and returns it in a user space in the format specified in the parameter list. The following discusses how the list is sorted for nonbatch jobs and for batch jobs.

When the job whose messages are being listed is not a batch job, the returned messages are sorted by their sending date and time unless the message being listed is a reply message to an inquiry, a sender's copy, or a notify message. If it is a reply message, it is listed immediately following the inquiry, sender's copy, or notify message with which it is associated.

If the last message listed is an inquiry, a sender's copy, or a notify message, the user of the API must call the API again. Use parameters that would include listing the next later message after the inquiry, sender's copy, or notify message in order to obtain an available reply message.

When the job whose messages are listed is a batch job, the messages are grouped into two categories:

- Request messages that have been or are being processed, and the other messages that occurred during the processing of those requests.
- Request messages that are yet to be processed, and any diagnostic messages associated with these request messages.

The API treats unprocessed request messages as if they had a sending time later than all the request messages and their associated messages that have been or are being processed. The following two examples describe the sorting further.

For example, if the call to this API specifies to list the messages for a batch job from oldest to newest, the list consists of all requests and their associated messages that have been or are being processed. They are sorted as described above for a job that is not a batch job. They are followed by any request messages and any associated diagnostic messages that have not yet been processed (in the order that they will be processed).

As an opposite example, if the call to this API specifies to list the messages for a batch job from newest to oldest, the list consists of the request messages that remain to be processed. They are in the opposite order that they will be processed. They are followed by the request messages that have been or are being processed and their associated messages. These are sorted backward through time as described above for nonbatch jobs.

The generated list replaces any existing lists in the user space.

If the user space is not large enough to contain the data to be returned, the user space is increased to the maximum user space size allowed (16MB) or the maximum amount of storage allowed to the user of the API. If this is not large enough to contain the data to be returned, only the number of complete messages that fits in the user space is returned. The information status field in the generic header is set to P (partial but accurate). The user can then resubmit the request from the last message returned to obtain the additional messages. The key of the last message listed for each message queue is provided in the ending message key field in the header portion of the user space.

The maximum messages requested field and the number of fields to return field for each listed message increase the system resources required to create the list. Users of this

API should use caution when specifying parameters that list many messages or request many identified fields to be returned for each listed message.

Authorities and Locks

User space *CHANGE

User space library
*USE

User space lock
*EXCLRD

Job authority

- *JOBCTL if the job for which messages are being retrieved has a user profile different from that of the job that calls the QMHLJOBL API.
- *ALLOBJ and *JOBCTL if the job for which messages are being retrieved has a user class of *SECOFR.

For additional information on job authorities, see the *Security Reference*.

Required Parameter Group

Qualified user space 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 user space library. 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 returned message information. You must use this format:

LJOB0100 Basic message information with identified return fields. This format is described in "LJOB0100 Format" on page 40-9.

Message selection information
INPUT; CHAR(*)
The information that determines the job message queue and messages to be listed. The format of this information is described in "JSLT0100 Format" on page 40-9

Size of message selection information
INPUT; BINARY(4)
The size in bytes of the message selection information parameter.

Format of message selection information
INPUT; CHAR(8)
The format of the message selection information parameter. You must use this format:

JSLT0100 The specific information identifying the messages to be listed. This format is described in "JSLT0100 Format" on page 40-9.

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 Lists

The user space created consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- LJOB0100 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 descriptions of each field in the list returned, see "Field Descriptions" on page 40-10.

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library specified
20	14	CHAR(8)	Format name specified
28	1C	CHAR(8)	Format of message selection information specified
36	24	BINARY(4)	Size of message selection information specified
40	28	BINARY(4)	Maximum messages requested specified
44	2C	CHAR(10)	List direction specified
54	36	CHAR(10)	Job name specified
64	40	CHAR(10)	User profile specified
74	4A	CHAR(6)	Job number specified
80	50	CHAR(16)	Internal job identifier specified
96	60	CHAR(4)	Starting message key specified
100	64	BINARY(4)	Maximum message length specified
104	68	BINARY(4)	Maximum message help length specified
108	6C	BINARY(4)	Offset to identifiers of fields to return specified
112	70	BINARY(4)	Number of fields to return specified

Offset		Type	Field
Dec	Hex		
116	74	BINARY(4)	Offset to call message queue specified
120	78	BINARY(4)	Length of call message queue specified
124	7C	CHAR(*)	Reserved
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of BINARY(4)	Identifiers of fields to return specified
		CHAR(*)	Call message queue specified

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library used
20	14	CHAR(4)	Starting message key used
24	18	CHAR(4)	Ending message key
28	1C	CHAR(10)	Job name used
38	26	CHAR(10)	User profile used
48	30	CHAR(6)	Job number used

LJOB0100 Format: The following table shows the information returned in the list data section of the user space for the LJOB0100 format. The offsets listed are from the beginning of the user space. For a detailed description of each field, see "Field Descriptions" on page 40-10.

The structure defined by this format is repeated for each message entry returned.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Offset to the next entry
4	4	BINARY(4)	Offset to fields returned
8	8	BINARY(4)	Number of fields returned
12	C	BINARY(4)	Message severity
16	10	CHAR(7)	Message identifier
23	17	CHAR(2)	Message type
25	19	CHAR(4)	Message key
29	1D	CHAR(10)	Message file name
39	27	CHAR(10)	Message file library specified at send time
49	31	CHAR(7)	Date sent
56	38	CHAR(6)	Time sent
62	3E	CHAR(*)	Reserved

Offset		Type	Field
Dec	Hex		
These fields repeat for each identifier field specified.		BINARY(4)	Offset to the next field information returned
		BINARY(4)	Length of field information returned
		BINARY(4)	Identifier field
		CHAR(1)	Type of data
		CHAR(1)	Status of data
		CHAR(14)	Reserved
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved

JSLT0100 Format

The following table lists the fields returned in the message selection information parameter.

The organization of the JSLT0100 format of the message selection information parameter follows. For a detailed description of each field, see "Field Descriptions" on page 40-10.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Maximum messages requested
4	4	CHAR(10)	List direction
14	E	CHAR(26)	Qualified job name
40	28	CHAR(16)	Internal job identifier
56	38	CHAR(4)	Starting message key
60	3C	BINARY(4)	Maximum message length
64	40	BINARY(4)	Maximum message help length
68	44	BINARY(4)	Offset to identifiers of fields to return
72	48	BINARY(4)	Number of fields to return
76	4C	BINARY(4)	Offset to call message queue name
80	50	BINARY(4)	Length of call message queue name
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of BINARY(4)	Identifiers of fields to return
		CHAR(*)	Call message queue name

Valid Field Identifiers The following table contains a list of the valid identifiers that can be specified in the message selection information parameter. For a detailed description of each field, see "Field Descriptions" on page 40-10.

Identifier	Type	Description
0101	CHAR(9)	Alert option
0201	CHAR(*)	Message data or text
0301	CHAR(*)	Message
0302	CHAR(*)	Message with substitution text
0401	CHAR(*)	Message help
0402	CHAR(*)	Message help with substitution text
0403	CHAR(*)	Message help with formatting characters
0404	CHAR(*)	Message help with substitution text and formatting characters
0501	CHAR(*)	Default reply
0601	CHAR(26)	Qualified sender job name
0602	CHAR(1)	Sender type
0603	CHAR(12)	Sending program name
0604	CHAR(10)	Sending module name
0605	CHAR(256)	Sending procedure name
0606	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers
0702	CHAR(1)	Receiving type
0703	CHAR(10)	Receiving program name
0704	CHAR(10)	Receiving module name
0705	CHAR(256)	Receiving procedure name
0706	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers or instruction numbers
0801	CHAR(10)	Message file library used
0901	CHAR(30)	Problem identification
1001	CHAR(10)	Reply status
1101	CHAR(10)	Request status
1201	BINARY(4)	Request level

Field Descriptions

Alert option. Whether and when an SNA alert is created and sent for the message. The value is one of the following:

**DEFER* An alert is sent after local problem analysis.

**IMMED* An alert is sent immediately when the message is sent to the history log (QHST).

**NO* No alert is sent.

**UNATTEND* An alert is sent immediately when the system is running in unattended mode (when the value of the alert status network attribute, ALRSTS, is *UNATTEND).

For more information about alerts, see the *Alerts and DSNX Guide*.

This field is set to blanks if no alert option was specified when the message was sent.

Call message queue name. The name of the call message queue from which the messages are listed. You must use one of these values:

* Messages from every call stack entry of the job are listed. If the length of call message queue name field is greater than 1, the value must be left-justified and padded on the right with blanks.

**EXT* Only messages sent to the external message queue (*EXT) of the job are to be listed. If the length of the call message queue name field is greater than 4. The value must be left-justified in the field and padded on the right with blanks.

Call message queue specified. The call message queue name as specified on the call to the API.

Data. The data returned for the specified identifier field.

Date sent. The date on which the message was sent, in CYYMMDD (century, year, month, and day) format.

Default reply. The text of the default reply when a stored message is being listed and a default reply exists. If this is not an inquiry message, or no default reply exists, this field is not used and the length of data field is 0.

Ending message key. The message key of the last message actually listed by the API. If no message is listed, the value returned is the same as the value specified in the starting message key specified field.

Format of message selection information specified. The format name of the message selection information parameter as specified on the call to the API.

Format name specified. The format name as specified on the call to the API.

Identifier field. The field ID actually returned in the LJOB0100 format. See "Valid Field Identifiers" for the list of valid field identifiers.

Identifiers of fields to return. The list of the field identifiers returned in the LJOB0100 format. For a list of the valid field identifiers, see "Valid Field Identifiers."

Identifiers of fields to return specified. The list of identifiers of fields to return as specified on the call to the API.

Internal job identifier. The internal name for the job. The List Job (QUSLJOB) API creates this identifier. If you do not specify *INT for the qualified job name parameter, this

parameter must contain blanks. If your application already has this information available from QUSLJOB, the QMHLJOBL API can locate the job more quickly with this information than with a job name. However, calling QUSLJOB solely to obtain this parameter for use by QMHLJOBL would result in poorer performance than using a job name in calling QMHLJOBL.

Internal job identifier specified. The internal job identifier as specified on the call to the API.

Job name specified. The job name of the job that lists the messages as specified on the call to the API.

Job name used. The actual job name of the job that was used to list the messages.

Job number specified. The job number of the job that lists the messages as specified on the call to the API.

Job number used. The actual job number of the job that was used to list the messages.

Length of data. The length of the data returned for the data field, in bytes.

Length of field information returned. The total length of information returned for this field, in bytes.

Length of call message queue name. The length of the call message queue name field, in bytes. The maximum length that can be specified is 256. The minimum length is 1.

Length of call message queue specified. The length of the call message queue specified field, in bytes.

List direction. The direction to list messages. You must use one of these directions:

**NEXT* Returns messages that are newer than the message specified by the starting message key field.

**PRV* Returns messages that are older than the messages specified by the starting message key field.

When a batch job is being listed, request messages that have not yet been processed or received are considered to have a sending date and time later than all other messages on the job log. This is also true for any diagnostic messages associated with those request messages.

List direction specified. The direction to list messages as specified on the call to the API.

Maximum message help length. The maximum number of characters of text that this API returns for field identifiers 0401, 0402, 0403, and 0404. (See "Valid Field Identifiers" on page 40-10.)

This value is not checked if field identifiers 0401, 0402, 0403 or 0404 are not specified. Specify a value to limit the number of characters returned for field identifiers 0401, 0402, 0403, and 0404. This value can be no smaller than 4. The maximum allowed value is 32765. To specify that the maximum length be used, use the special value of -1.

Maximum message help length specified. The maximum number of characters to return for field identifiers 0401, 0402, 0403 and 0404 as specified on the call to the API.

Maximum message length. The maximum number of characters of text that this API returns for field identifiers 0301 and 0302.

This value is not checked if field identifiers 0301 or 0302 are not specified. Specify a value to limit the number of characters returned for field identifiers 0301 and 0302. (See "Valid Field Identifiers" on page 40-10.) This value can be no smaller than 4. The maximum allowed value is 32765. To specify that the maximum length be used, use the special value -1.

Maximum message length specified. The maximum number of characters to return for field identifiers 0301 and 0302 as specified on the call to the API.

Maximum messages requested. The maximum number of messages to be returned.

If fewer messages than the number requested exist on the job message queue, only the number of messages that exist are returned. No error is signaled, and the information status field in the generic header would be marked as C for complete and accurate.

To list all messages in the job log in the specified list direction from the starting message key, use the special value of -1.

Maximum messages requested specified. The number of messages requested to be listed as specified on the call to the API.

Message. The text of a predefined message without message data substitution. If an immediate message is listed, this field contains the immediate message text.

Message data or text. The values for substitution variables in a predefined message, or the text of an immediate message. If the message identifier field is not blank, this field contains message data. If the message identifier field is blank, this field contains immediate message text.

Any pointer data in this field is marked as not valid if both:

- The API is called by a call stack entry that is not in system state.
- The system security level is 50 or above.

Message file library specified at send time. The name of the library containing the message file as specified when the message was sent. If *CURLIB or *LIBL were specified for the library when the message was sent, 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 actual name of the library that contains the message file used to retrieve the message information. If an immediate message is listed, this field is set to blanks.

Message file name. The name of the message file containing the message listed.

Message help. The message help for the message listed without formatting characters and without substitution of data. If an immediate message is listed, this field contains the immediate message text.

Message help with formatting characters. The message help for the message listed, including formatting characters.

Three format control characters can be returned within the message. They are defined in the Add Message Description (ADDMSGD) command of the *CL Reference* to have these meanings:

&N Forces the text to a new line (column 2). If the text is longer than one line, the next lines are indented to column 4 until the end of text or another format control character is found.

&P Forces the text to a new line indented to column 6. If the text is longer than one line, the next lines start in column 4 until the end of text or another format control character is found.

&B Forces the text to a new line, starting in column 4. If the text is longer than one line, the next lines are indented to column 6 until the end of text or another format control character is found.

If an immediate message is listed, this field contains the immediate message text.

Message help with substitution text. The message help for the message listed, including the message data. If an immediate message is listed, this field contains the immediate message text.

Message help with substitution text and formatting characters. The message help for the message listed, including the substitution text and the formatting characters. See the description of the message help with formatting characters field for an explanation of formatting characters. If an immediate message is listed, this field contains the immediate message text.

Message identifier. The identifying code of the message listed. If an immediate message is listed, this field is set to blanks.

Message key. The message reference key of the message listed.

Message severity. The severity of the message listed. Possible values are 0 through 99.

Message type. The type of message listed. The possible values and their meanings follow:

Value	Message Type
01	Completion
02	Diagnostic
04	Informational
05	Inquiry
06	Sender's copy
08	Request
10	Request with prompting
14	Notify, exception already handled when API is called
15	Escape, exception already handled when API is called
16	Notify, exception not handled when API is called
17	Escape, exception not handled when API is called
21	Reply, not checked for validity
22	Reply, checked for validity
23	Reply, message default used
24	Reply, system default used
25	Reply, from system reply list

Message with substitution text. The text of a predefined message with the message data included. If an immediate message is listed, this field contains the immediate message text.

Number of fields returned. The number of identifier fields returned to the application.

Number of fields to return. The number of fields to return in the LJOB0100 format (the number of entries in the identifiers of fields to return array).

Number of fields to return specified. The number of identifier fields to return as specified on the call to the API.

Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers The number of statement numbers or instruction numbers available for the receiving program or procedure.

For OPM programs and nonoptimized procedures, this count is 1.

For optimized procedures, this count can be greater than 1. In this case, each statement number represents a potential point at which the message could have been received. If the mapping table information has been removed from the program, this field returns a count of 0 and no statement numbers are available. The array of receiving statement numbers or instruction numbers immediately follows this field in the returned data.

Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers. The number of statement numbers or instruction numbers available for the sending program or procedure. For OPM programs and nonoptimized procedures, this count is 1. For optimized procedures, this count can be greater than 1. In this case, each

| statement number represents a potential point at which the
| message could have been sent. If the mapping table infor-
| mation has been removed from the program, this field returns
| a count of 0, and no statement numbers are available. The
| array of sending statement numbers or instruction numbers
| immediately follows this field in the returned data.

| **Offset of fields to return specified**. The offset, in bytes,
| from the beginning of the user space to the beginning of the
| identifiers of fields to return specified field.

| **Offset to call message queue name**. The offset in bytes
| from the beginning of the message selection information
| parameter to the beginning of the call message queue name
| field.

| **Offset to call message queue specified**. The offset,in
| bytes, from the beginning of the user space to the beginning
| of the call message queue specified field.

| **Offset to fields returned**. The offset, in bytes, from the
| beginning of the user space to the beginning of the first
| repeating identified field of the LJOB0100 format.

| **Offset to identifiers of fields to return**. The offset in bytes
| from the beginning of the message selection information
| parameter to the beginning of the identifiers of fields to return
| array.

| **Offset to the next entry**. The offset from the beginning of
| the user space to the beginning of the next message entry,
| in bytes.

| **Offset to the next field information returned**. The offset
| from the beginning of the user space to the beginning of the
| next repeating identified field of the LJOB0100 format, in
| bytes.

| **Problem identification**. This field can be specified for the
| QMHLJOBL API, but it never returns any data and the length
| of data field is 0.

| **Qualified job name**. The name of the job whose messages
| are to be listed. 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
| rest of the qualified job name parameter must
| be blank.
| *User name*
| CHAR(10)
| A specific user profile name or blanks when the job
| name is the special value * or *INT.

| *Job number*
| CHAR(6)
| A specific job number or blanks when the job name is
| the special value * or *INT.

| **Qualified sender job name**. This field can be specified for
| the QMHLJOBL API, but it never returns any data and the
| length of data field is 0.

| **Receiving module name**. The name of the module that
| contains the procedure where the message was sent. If the
| message was not sent to a procedure within a ILE program,
| this field is not used and the length of data field is 0.

| **Receiving procedure name**. The name of the procedure
| receiving the message. If the message was not sent to a
| procedure within an ILE program, this field is not used and
| the length of data field is 0.

| **Receiving program name**. The program name, or the ILE
| program name that contains the procedure that the message
| was sent to.

| **Receiving type**. The type of the receiver (whether it is a
| program or a procedure). Possible values and their
| meanings follow:

| *0* Receiver is a program
| *1* Receiver is a procedure within an ILE program

| **Reply status**. The reply status of the message (whether it
| accepts a reply, and if so, whether a reply has been sent).
| Possible values and their meanings follow:

| *A* Message accepts a reply, and a reply has been sent.
| *W* Message accepts a reply, and a reply has not been
| sent. (The message is waiting for a reply.)
| *N* Message does not accept a reply.

| **Request level**. The level of the request-processing program
| that received the request message. If the message being
| listed is not a request, this field is set to 0.

| **Request status**. Information regarding the processing
| status of the request message. Possible values and their
| meanings follow:

| *O* This request message has been received and pro-
| cessed.
| *C* This request message is currently being processed.
| *N* This request message has not yet been processed.

| If the message being listed is not a request, this field is set
| to a blank character.

| **Reserved**. An ignored field.

| **Sending type**. The type of the sender (whether it is a
| program or procedure). Possible values and their meanings
| follow:

| *0* Sender is a program
| *1* Sender is a procedure within an ILE program

Sending module name. The name of the module that contains the procedure sending the message. If the message was not sent by a procedure within an ILE program, this field is not used and the length of data field is 0.

Sending procedure name. The name of the procedure sending the message. If the message was not sent by a procedure within an ILE program, this field is not used and the length of data field is 0.

Sending program name. The sending program name or ILE program name that contains the procedure sending the message. Under certain conditions, the actual name of the program that sent the message is not known. In these cases, this field contains the 6-byte hexadecimal address of the program converted into 12 displayable characters. In all other cases, the 10-character program name is returned left-justified in the field. The final 2 characters contain blanks.

Size of message selection information specified. The size of the message selection information field, in bytes, as specified in the call to the API.

Starting message key. The message key to begin searching for messages to list from the job.

You can use these special values for the message key:

'00000000'X The first message to be returned is the oldest message in the queue.

'FFFFFFFF'X The first message to be returned is the newest message in the queue.

When the list direction field is *NEXT, the first message listed is the first message with a message key equal to or greater than the key specified. If no message is found in this manner, an error is returned. When the list direction field is *PRV, the first message listed is the first message with a message key equal or less than the key specified. If no message is found in this manner, an error is returned.

If a key of a reply message is specified, the message search begins with the inquiry, sender's copy, or notify message with which the reply is associated. It does not begin with the reply message itself.

When the call message queue name field is *EXT, and the message specified by the starting message key exists but is not on the *EXT message queue, an error is returned.

Note: When a batch job is being listed, request messages that have not yet been processed or received are considered to have a sending date and time later than all other messages on the job log. This is also true for any diagnostic messages associated with those request messages. If the starting message key provided is to one of these messages, and the list direction is *NEXT, only additional request messages and associated diagnostic messages that remain to be processed are listed. If the list direction is *PRV, any earlier request messages or associated diagnostic messages are listed. These are followed by messages on the job log associated with request messages that have been or are being processed.

Starting message key specified. The starting message key as specified on the call to the API.

Starting message key used. The message keys of the first message actually listed by the API. If no message is listed, the value returned is the same as the value specified in the starting message key specified field.

Status of data. The status of the data listed for this message. Possible values and their meanings follow:

blank The data returned is complete.

A The caller of the API was not authorized to view the data. This occurs when the caller of the API is not authorized to the message file or the message file library containing a stored message being listed.

D The data was damaged. This occurs when the message file or library specified at send time for a stored message is damaged when the API is called.

U The data was unavailable. This occurs when the message file or library specified at send time for a stored message is exclusively used by another process when the API is called.

N The data was not found. This occurs when the message file or library specified at send time for a stored message cannot be found or resolved when the API is called.

This field is applicable to the field identifiers that are retrieved from the message file for a stored message. A description of the action that occurs for specific field identifiers when the status of data field is not blank follows:

0101 When the status of data field is not blank, the alert option field identifier contains blanks.

0301, 0302 When the status of data field is not blank, these message field identifiers contain message text. The text of the message regards the problem encountered while attempting to access the message file. Both fields have the message data substituted.

0401, 0402, 0403, 0404 When the status of data field is not blank, these message help field identifiers contain the text of the message regarding the problem encountered while attempting to access the message file. All fields have the message data substituted. The message help with formatting characters and message help with substitution text and formatting characters field identifiers also have the message formatting characters included.

0501 When the status of data field is not blank, the default reply field identifier contains the system default reply.

0801 When the status of data field is not blank, the message file library used field identifier contains blanks.

This field is also applicable to the various sending and receiving information fields when a problem is encountered while attempting to retrieve this information. This includes field identifiers 0602, 0603, 0604, 0605, 0606, 0702, 0703, 0704, 0705, and 0706. When one of these fields cannot be

retrieved from the message, the status of data field is set to U and the field is set to blanks.

The status of data field is always blank for the other field identifiers.

Time sent. The time at which the message being listed was sent, in HHMMSS (hour, minute, and second) format.

Type of data. The type of data returned.

C The data is returned in character format.
B The data is returned in binary format.
M The data is returned in a mixed format. This value is returned for the field IDs 0606 and 0706, which contain a binary(4) value followed by an array of 10-character elements.

User profile specified. The user profile of the job from which to list messages as specified on the call to the API.

User profile used. The actual user profile of the job used from which to list messages.

User space library specified. The name of the user space library as specified on the call to the API.

User space library used. The actual name of the library where the user space was found.

User space name specified. The name of the user space as specified on the call to the API.

User space name used. The actual name of the user space used to store the data listed.

Error Messages

CPF1866 E Value &1 for number of fields to return not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B7 E Value &1 for call stack entry name length not valid.
CPF240D E Message search direction specified is not valid.
CPF240E E Format name of message selection information is not valid.
CPF240F E Field identifier is not valid or is a duplicate of another field indentifier specified.
CPF241E E Call stack entry name is not valid.
CPF241F E Length &1 specified for maximum message length is not valid.
CPF2410 E Message key not found in message queue &1.
CPF2441 E Not authorized to display job log.
CPF2443 E Job log not displayed or listed because job has ended.
CPF247D E Size of message selection information, &1, is not valid.
CPF2476 E The maximum number of messages to list, &1, is not valid.
CPF252F E Length &1 specified for maximum message help length is not valid.
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.
CPF3C51 E Internal job identifier not valid.
CPF3C52 E Internal job identifier no longer valid.
CPF3C53 E Job &3/&2/&1 not found.
CPF3C55 E Job &3/&2/&1 does not exist.
CPF3C58 E Job name specified is not valid
CPF3C59 E Internal identifier is not blanks and job name is not *INT.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9811 E Program &1 in library &2 not found.
CPF9812 E File &1 in library &2 not found.
CPF9814 E Device &1 not found.
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.
CPF9831 E Cannot assign device &1.
CPF9838 E User profile storage limit exceeded.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Nonprogram Messages (QMHLSTM) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Message selection information	Input	Char(*)
4	Size of message selection information	Input	Binary(4)
5	Format of message selection information	Input	Char(8)
6	Error code	I/O	Char(*)

The List Nonprogram Messages (QMHLSTM) API lists messages from one or two nonprogram message queues. This API obtains the requested message information and returns it in a user space through the format specified in the parameter list.

If multiple message queues are specified, the messages are sorted by their sending dates and times. However, messages listed as a reply message to an inquiry or a sender's copy message are sorted differently. When a reply message exists for one of these types, it is listed immediately following the inquiry or sender's copy message with which it is associated.

If messages in multiple queues have identical sending dates and times, the message from the first queue in the qualified message queue names array is listed first.

If the last message listed is an inquiry or sender's copy message, the user must call the API again. Use parameters

that would include listing the next later message after the inquiry or sender's copy message in order to determine and obtain an available reply message.

The QMHLSTM API only lists messages from nonprogram message queues. The QMHLSTM API cannot be used to list messages sent to a job log (including the external message (*EXT) queue of a job). See "List Job Log Messages (QMHLJOBL) API" on page 40-7. QMHLSTM cannot be used to list messages sent to the history log (QHST).

The generated list replaces any existing lists in the user space.

If the user space is not large enough to contain the amount of data that is to be returned, the user space is increased. It is increased to the maximum user space size allowed (16MB) or to the maximum amount of storage allowed to the user of the API. If this is not large enough to contain the data to be returned, only the number of complete messages that fit in the user space is returned. The information status field in the generic header is set to P (partial but accurate). The user can then resubmit the request from the last message returned to obtain the additional messages. The key of the last message listed for each message queue is provided in the ending message key field in the header portion of the user space.

New messages are prevented from being added to or removed from the message queues being listed during the use of this API. The maximum messages requested field and the number of fields to return field for each listed message increases the time that the queues are unavailable. Users of this API should use caution when listing many messages or many fields. Try to avoid causing lock-out situations on highly used message queues.

Authorities and Locks

Message queue	*USE
Message queue library	*READ
User space	*CHANGE
User space library	*USE
User space lock	*EXCLRD

Required Parameter Group

Qualified user space 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 user space library.

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 returned message information. You must use this format:

LSTM0100 Basic message information with identified return fields. This format is described in "LSTM0100 Format" on page 40-17.

Message selection information
INPUT; CHAR(*)
The information that determines the message queues and messages to be listed. The format of this information is described in "MSLT0100 Format:" on page 40-17.

Size of message selection information
INPUT; BINARY(4)
The size, in bytes, of the message selection information parameter.

Format of message selection information
INPUT; CHAR(8)
The format of the message selection information parameter. You must use this format:

MSLT0100 The specific information identifying the messages to be listed. This format is described in "MSLT0100 Format:" on page 40-17.

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 Lists

The user space created consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- LSTM0100 format
- MSLT0100 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 descriptions of each field in the list returned, see "Field Descriptions" on page 40-18.

Input Parameter Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library specified
20	14	CHAR(8)	Format name specified

Offset		Type	Field
Dec	Hex		
28	1C	CHAR(8)	Format of message selection information specified
36	24	BINARY(4)	Size of message selection information specified
40	28	BINARY(4)	Maximum messages requested specified
44	2C	CHAR(10)	List direction specified
54	36	CHAR(10)	Selection criteria specified
64	40	BINARY(4)	Severity criteria specified
68	44	BINARY(4)	Maximum message length specified
72	48	BINARY(4)	Maximum message help length specified
76	4C	BINARY(4)	Offset to message queue names specified
80	50	BINARY(4)	Offset to starting message keys specified
84	54	BINARY(4)	Number of message queues specified
88	58	BINARY(4)	Offset to identifiers of fields to return specified
92	5C	BINARY(4)	Number of fields to return specified
96	60	CHAR(*)	Reserved
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of CHAR(20)	Message queue names specified
		ARRAY(*) of CHAR(4)	Starting message key specified
		ARRAY(*) of BINARY(4)	Identifiers of fields to return specified

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library used
20	14	BINARY(4)	Offset to message queue names used
24	18	BINARY(4)	Offset to starting message keys used
28	1C	BINARY(4)	Offset to ending message keys
32	20	BINARY(4)	Number of message queues used
36	24	CHAR(*)	Reserved
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of CHAR(20)	Message queue names used
		ARRAY(*) of CHAR(4)	Starting message key used
		ARRAY(*) of CHAR(4)	Ending message key used

LSTM0100 Format: The following table shows the information returned in the list data section of the user space for the LSTM0100 format. The offsets listed are from the beginning of the user space. For a detailed description of each field, see "Field Descriptions" on page 40-18.

The structure defined by this format is repeated for each message entry returned.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Offset to the next entry
4	4	BINARY(4)	Offset to fields returned
8	8	BINARY(4)	Number of fields returned
12	C	BINARY(4)	Message severity
16	10	CHAR(7)	Message identifier
23	17	CHAR(2)	Message type
25	19	CHAR(4)	Message key
29	1D	CHAR(10)	Message file name
39	27	CHAR(10)	Message file library specified at send time
49	31	CHAR(10)	Message queue
59	3B	CHAR(10)	Message queue library used
69	45	CHAR(7)	Date sent
76	4C	CHAR(6)	Time sent
82	52	CHAR(*)	Reserved
These fields repeat for each identifier field specified.		BINARY(4)	Offset to the next field information returned
		BINARY(4)	Length of field information returned
		BINARY(4)	Identifier field
		CHAR(1)	Type of data
		CHAR(1)	Status of data
		CHAR(14)	Reserved
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved

MSLT0100 Format: The organization of the MSLT0100 format of the message selection information parameter follows. For a detailed description of each field, see "Field Descriptions" on page 40-18.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Maximum messages requested
4	4	CHAR(10)	List direction
14	E	CHAR(10)	Selection criteria
24	18	BINARY(4)	Severity criteria
28	1C	BINARY(4)	Maximum message length
32	20	BINARY(4)	Maximum message help length
36	24	BINARY(4)	Offset to qualified message queue names
40	28	BINARY(4)	Offset to starting message keys
44	2C	BINARY(4)	Number of message queues
48	30	BINARY(4)	Offset to identifiers of fields to return
52	34	BINARY(4)	Number of fields to return
The offsets to these fields are specified in the previous offset variables.		ARRAY(*) of CHAR(20)	Qualified message queue names
		ARRAY(*) of CHAR(4)	Starting message key
		ARRAY(*) of BINARY(4)	Identifiers of fields to return

Valid Field Identifiers The following table contains a list of the valid identifiers for the LSTM0100 format and the MSLT0100 format.

Identifier	Type	Description
0101	CHAR(9)	Alert option
0201	CHAR(*)	Message data or text
0301	CHAR(*)	Message
0302	CHAR(*)	Message with substitution text
0401	CHAR(*)	Message help
0402	CHAR(*)	Message help with substitution text
0403	CHAR(*)	Message help with formatting characters
0404	CHAR(*)	Message help with substitution text and formatting characters
0501	CHAR(*)	Default reply
0601	CHAR(26)	Qualified sender job name
0602	CHAR(1)	Sender type
0603	CHAR(12)	Sending program name
0604	CHAR(10)	Sending module name
0605	CHAR(256)	Sending procedure name
0606	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers
0702	CHAR(1)	Receiving type
0703	CHAR(10)	Receiving program name
0704	CHAR(10)	Receiving module name
0705	CHAR(256)	Receiving procedure name
0706	BINARY(4) followed by ARRAY(*) of CHAR(10)	Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers or instruction numbers
0801	CHAR(10)	Message file library used
0901	CHAR(30)	Problem identification
1001	CHAR(10)	Reply status
1101	CHAR(10)	Request status
1201	BINARY(4)	Request level

Field Descriptions

Alert option. Whether and when an SNA alert is created and sent for the message. If a message is listed, the value is one of the following:

**DEFER* An alert is sent after local problem analysis.
**IMMED* An alert is sent immediately when the message is sent to the history log (QHST).
**NO* No alert is sent.
**UNATTEND*
An alert is sent immediately when the system is running in unattended mode (when the value of the alert status network attribute, ALRSTS, is *UNATTEND).

For more information on alerts, see the *Alerts and DSNX Guide*.

This field is set to blanks if no alert option was specified when the message was sent.

Data. The data returned for the specified identifier field.

Date sent. The date on which the message was sent, in CYYMMDD (century, year, month, and day) format.

Default reply. The text of the default reply. When a stored message is being listed and a default reply exists. If this is not an inquiry message or no default reply exists, this field is not used and the length of data field is 0.

Ending message key used. The message keys of the last message actually listed by the API. If no message is listed from a particular queue, the value returned for that queue is the same as that in the starting message key specified field.

| **Format of message selection information specified**. The format name of the message selection information parameter as specified on the call to the API.

| **Format name specified**. The format name as specified on the call to the API.

| **Identifier field**. The field returned. See "Valid Field Identifiers" on page 40-18 for the list of valid field identifiers.

| **Identifiers of fields to return**. The list of the field identifiers to be returned in the LSTM0100 format. For a list of the valid field identifiers, see "Valid Field Identifiers" on page 40-18. An error is returned if the identifier matches one specified earlier in the array, or if the identifier is not valid.

| **Identifiers of fields to return specified**. The list of field identifiers to return as specified on the call to the API.

| **Length of data**. The length of the data returned for the data field, in bytes.

| **Length of field information returned**. The total length of information returned for this field, in bytes.

| **List direction**. The direction to list messages. You must use one of these directions:

| **NEXT* Returns messages that are newer than the messages specified by the starting message key field.

| **PRV* Returns messages that are older than the message specified by the starting message key field.

| If multiple message queues are to be listed, messages are intermixed and sorted by date and time in the specified manner.

| **List direction specified**. The direction to list messages as specified on the call to the API.

| **Maximum message help length**. The maximum number of characters of text that this API returns for field identifiers 0401, 0402, 0403, and 0404. (See "Valid Field Identifiers" on page 40-18.)

| This value is not checked if field identifiers 0401, 0402, 0403, or 0404 are not specified. Specify a value to limit the number of characters returned for field identifiers 0401, 0402, 0403, and 0404. This value can be no smaller than 4. The maximum allowed value is 32765. To specify that the maximum length be used, use the special value of -1.

| **Maximum message help length specified**. The maximum number of characters to return for field identifiers 0401, 0402, 0403, and 0404 as specified on the call to the API.

| **Maximum message length**. The maximum number of characters of text that this API returns for field identifiers 0301 and 0302. (See "Valid Field Identifiers" on page 40-18.)

| This value is not checked if field identifiers 0301 or 0302 are not specified. Specify a value to limit the number of characters returned for field identifiers 0301 and 0302. This value can be no smaller than 4. The maximum allowed value is 32765. To specify that the maximum length be used, use the special value of -1.

| **Maximum message length specified**. The maximum number of characters to return for field identifiers 0301 and 0302 as specified on the call to the API.

| **Maximum messages requested**. The maximum number of messages to be returned.

| If fewer messages than the number requested exist on the queues, only the number of messages that exist are returned. No error is signaled, and the information status field in the generic header would be marked as C for complete and accurate.

| Use the special value of -1 to list all messages on the queues in the specified list direction. The list runs from the starting message keys that meet the selection and severity criteria.

| **Maximum messages requested specified**. The number of messages requested to be listed as specified on the call to the API.

| **Message**. The text of a predefined message without message data substitution. If an immediate message is listed, this field contains the immediate message text.

| **Message data or text**. The values for substitution variables in a predefined message, or the text of an immediate message. If the message identifier field is not blank, this field contains message data. If the message identifier field is blank, this field contains immediate message text.

| Any pointer data in this field is marked as not valid if both:

| - The API is called by a call stack entry that is not in system state.
| - The system security level is 50 or above.

| **Message file library specified at send time**. The name of the library containing the message file as specified when the message was sent. If *CURLIB or *LIBL was specified for the library when the message was sent, 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 actual name of the library that contains the message file used to retrieve the message information. If an immediate message is listed, this field is set to blanks.

| **Message file name**. The name of the message file containing the message listed.

| **Message help**. The message help for the message listed without formatting characters and without substitution of data.

If an immediate message is listed, this field contains the immediate message text.

Message help with formatting characters. The message help for the message listed, including formatting characters.

Three format control characters can be returned within the message. They are defined in the Add Message Description (ADDMSGD) command of the *CL Reference* to have these meanings:

&N Forces the text to a new line (column 2). If the text is longer than one line, the next lines are indented to column 4 until the end of text or another format control character is found.

&P Forces the text to a new line indented to column 6. If the text is longer than one line, the next lines start in column 4 until the end of text or another format control character is found.

&B Forces the text to a new line, starting in column 4. If the text is longer than one line, the next lines are indented to column 6 until the end of text or another format control character is found.

If an immediate message is listed, this field contains the immediate message text.

Message help with substitution text. The message help for the message listed, including the message data. If an immediate message is listed, this field contains the immediate message text.

Message help with substitution text and formatting characters. The message help for the message listed, including the substitution text and the formatting characters. See the message help with formatting characters field for an explanation of formatting characters. If an immediate message is listed, this field contains the immediate message text.

Message identifier. The identifying code of the message listed. If an immediate message is listed, this field is set to blanks.

Message key. The key of the message listed.

Message queue. The name of the message queue where the message was listed.

Message queue library used. The actual library that contains the message queue.

Message queue names specified. The qualified message queue names specified on the call to the API.

Message queue names used. The actual message queue names used to list messages. The first 10 characters are the message queue name, and the second 10 characters are the message queue library.

Message severity. The severity of the message listed. Possible values are 0 through 99.

Message type. The type of message listed. The possible values and their meanings follow:

Value	Message Type
01	Completion
02	Diagnostic
04	Informational
05	Inquiry
06	Sender's copy
08	Request
10	Request with prompting
14	Notify, exception already handled when API is called
15	Escape, exception already handled when API is called
16	Notify, exception not handled when API is called
17	Escape, exception not handled when API is called
21	Reply, not checked for validity
22	Reply, checked for validity
23	Reply, message default used
24	Reply, system default used
25	Reply, from system reply list

Message with substitution text. The text of a predefined message with the message data included. If an immediate message is listed, this field contains the immediate message text.

Number of fields returned. The number of identifier fields returned to the application.

Number of fields to return. The number of fields to return in the LSTM0100 format (the number of entries in the identifiers of fields to return array).

Number of fields to return specified. The number of identifier fields to return as specified on the call to the API.

Number of message queues. The number of message queues to list. The valid values follow:

1 One message queue listed. Both the qualified message queue names field and the starting message key field contain one entry.

2 Two message queues listed. Both the qualified message queue names field and the starting message key field contain two entries.

Number of message queues specified. The number of message queues to be listed as specified on the call to the API. This is the size of the declared array of the message queue names specified and the starting message key specified fields.

Number of message queues used. The number of message queues listed. This is the number of elements in the message queue names used, starting message key used, and ending message key arrays.

Number of sending statement numbers or instruction numbers available followed by an array of the sending statement numbers or instruction numbers This field can

be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Number of receiving statement numbers or instruction numbers available followed by an array of the receiving statement numbers or instruction numbers This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Offset to ending message key. The offset from the beginning of the user space to the beginning of the ending message key field, in bytes.

Offset to fields returned. The offset from the beginning of the user space to the beginning of the first repeating identified field of the LSTM0100 format, in bytes.

Offset to identifiers of fields to return. The offset in bytes from the beginning of the message selection information parameter to the beginning of the identifiers of fields to return field.

Offset to identifiers of fields to return specified. The offset from the beginning of the user space to the beginning of the identifiers of fields to return specified field, in bytes.

Offset to message queue names specified. The offset from the beginning of the user space to the beginning of the message queue names specified field, in bytes.

Offset to message queue names used. The offset from the beginning of the user space to the beginning of the message queue names used field, in bytes.

Offset to qualified message queue names. The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the qualified message queue names field.

Offset to starting message keys. The offset, in bytes, from the beginning of the message selection information parameter to the beginning of the starting message key field.

Offset to starting message key specified. The offset from the beginning of the user space to the beginning of the starting message key specified field, in bytes.

Offset to starting message key used. The offset from the beginning of the user space to the beginning of the starting message key used field, in bytes.

Offset to the next entry. The offset from the beginning of the user space to the beginning of the next message entry, in bytes.

Offset to the next field information returned. The offset from the beginning of the user space to the beginning of the next repeating identified field of the LSTM0100 format, in bytes.

Problem identification. The number the system generates to identify a problem if problem analysis can be run for the message being listed. The problem identification is in the following format:

CHAR 1-10 Problem ID number. The number the system generates to identify the problem.
CHAR 11-30 Origin system in the format *network-ID.control-point-name.*

If problem analysis cannot be run, and this field is specified, the length of data field is 0.

Qualified message queue names. The list of message queues and the libraries where the message queues are located. The number of entries in the array must match the number specified in the number of message queues field. The first 10 characters of each entry contain the message queue name, and the second 10 characters of each entry contain the message queue library.

You can use the following values for the library name:

**CURLIB* The job's current library
**LIBL* The library list

Qualified sender job name. The name of the job that sent the message. The job name has three parts:

CHAR 1-10 The specific job name.
CHAR 11-20 The specific user profile name.
CHAR 21-26 The specific job number.

Receiving module name. This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Receiving procedure name. This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Receiving program name. This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Receiving type. This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Reply status. The reply status of the message (whether it accepts a reply, and if so, whether a reply has been sent). Possible values and their meanings follow:

A Message accepts a reply, and a reply has been sent.
W Message accepts a reply, and a reply has not been sent. (The message is waiting for a reply.)
N Message does not accept a reply.

Request level. This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Request status. This field can be specified for the QMHLSTM API, but it never returns any data the length of data field is 0.

Reserved. An ignored field.

Selection criteria. The type of messages to be listed. Valid values follow:

**ALL* All messages are to be listed.
**MNNR* Only messages not requiring a reply are listed. This includes informational, completion, diagnostic, request, notify, escape, reply, answered inquiry, and answered sender's copy messages.
**MNR* Only messages needing a reply are listed. This includes only unanswered inquiry messages.
**PAR* Only messages that can have problem analysis run against them are listed.
**SCNR* Only sender's copy messages requiring a reply are listed. This includes only unanswered sender's copy messages.

Selection criteria specified. The selection criteria as specified on the call to the API.

Sending type. This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Sending module name. This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Sending procedure name. This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Sending program name. The sending program name or ILE program name that contains the procedure sending the message. Under certain conditions, the actual name of the program that sent the message is not known. In these cases, this field contains the 6-byte hexadecimal address of the program converted into 12 displayable characters. In all other cases, the 10-character program name is returned left-justified in the field; the final 2 characters contain blanks.

Sending statement numbers or instruction numbers. This field can be specified for the QMHLSTM API, but it never returns any data and the length of data field is 0.

Severity criteria. The minimum severity of a message to be included in the list. The value must be between 0 and 99. To retrieve all messages, specify a severity criteria of 0.

Severity criteria specified. The severity criteria as specified on the call to the API.

Size of message selection information specified. The size of the message selection information field, in bytes, as specified in the call to the API.

Starting message key. The message key used to begin searching for messages to list from the corresponding entry in the qualified message queue names field.

You can use these special values for the message keys:

'00000000'X The first message to be returned is the oldest message in the queue.
'FFFFFFFF'X The first message to be returned is the newest message in the queue.

If a value other than X'00000000' or X'FFFFFFFF' is specified and a message with that key does not exist, an error is returned.

If the message specified by the starting message key exists but does not meet the selection criteria and severity criteria fields specified, no error is returned. The search for messages to list begins from the message specified by the starting message key.

Starting message key specified. The starting message keys as specified on the call to the API.

Starting message key used. The message keys of the first message actually listed by the API. If no message is listed from a particular message queue, the value returned for that queue is the same as that in the starting message key specified field.

Status of data. The status of the data listed for this message. Possible values and their meanings follow:

blank The data returned is complete.
A The caller of the API was not authorized to view the data. This occurs when the caller of the API is not authorized to the message file or message file library containing a stored message being listed.
D The data was damaged. This occurs when the message file or library specified at send time for a stored message is damaged when the API is called.
U The data was unavailable. This occurs when the message file or library specified at send time for a stored message is exclusively used by another process when the API is called.
N The data was not found. This occurs when the message file or library specified at send time for a stored message cannot be found or resolved when the API is called.

This field is applicable to the field identifiers that are retrieved from the message file for a stored message. A description of the action that occurs for specific field identifiers when the status of data field is not blank follows:

0101 When the status of data field is not blank, the alert option field identifier contains blanks.
0301, 0302 When the status of data field is not blank, these message field identifiers contain message text. The text of the message regards the problem encountered while attempting to access the message file. Both fields have the message data substituted.
0401, 0402, 0403, 0404 When the status of data field is not blank, these message help field identifiers contain the text of the message regarding the problem encountered while attempting to access the message file. All fields have the message data substituted. The message help with formatting characters and message help with sub-

stitution text and formatting characters field identifiers also have the message formatting characters included.

0501 When the status of data field is not blank, the default reply field identifier contains the system default reply.

0801 When the status of data field is not blank, the message file library used field identifier contains blanks.

This field is also applicable to the various sending information fields (identifiers 0601, 0603) when a problem is encountered while attempting to retrieve this information. When one of these fields cannot be retrieved from the message:

- The status of data field is set to N.
- The length of data field is set to 0.

The status of data field is always blank for the other field identifiers. The length of data field is zero.

Time sent. The time at which the message being listed was sent, in HHMMSS (hour, minute, and second) format.

Type of data. The type of data returned.

C The data is returned in character format.

B The data is returned in binary format.

M The data is returned in a mixed format. This value is returned for the field IDs 0606 and 0706.

User space library specified. The name of the user space library as specified on the call to the API.

User space library used. The actual name of the library where this user space was found.

User space name specified. The name of the user space as specified on the call to the API.

User space name used. The actual name of the user space used to store the data listed.

Error Messages

CPF1866 E Value &1 for number of fields to return not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF240D E Message search direction specified is not valid.
CPF240E E Format name of message selection information is not valid.
CPF240F E Field identifier is not valid or is a duplicate of another field identifier specified.
CPF2401 E Not authorized to library &1.
CPF2403 E Message queue &1 in &2 not found.
CPF2408 E Not authorized to message queue &1.
CPF241D E Severity criteria specified is not valid.
CPF241F E Length &1 specified for maximum message length is not valid.
CPF2410 E Message key not found in message queue &1.
CPF2433 E Function not allowed for system log message queue &1.
CPF2444 E Number of message queues, &1, is not valid.
CPF2467 E &3 message queue &1 in library &2 logically damaged.
CPF247D E Size of message selection information, &1, is not valid.
CPF2476 E The maximum number of messages to list, &1, is not valid.
CPF2477 E Message queue &1 currently in use.
CPF252F E Length &1 specified for maximum message help length is not valid.
CPF2538 E Value for selection criteria not valid.
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.
CPF8198 E Damaged object found.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9811 E Program &1 in library &2 not found.
CPF9812 E File &1 in library &2 not found.
CPF9814 E Device &1 not found.
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.
CPF9838 E User profile storage limit exceeded.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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	Call message queue	Input	Char(*)
5	Call stack counter	Input	Binary(4)
6	Error code	I/O	Char(*)

Optional Parameter Group:

7	Length of call message queue	Input	BINARY(4)
8	Call stack entry qualification	Input	CHAR(20)

The Move Program Message (QMHMOVPM) API moves messages from the current call message queue to the message queue of a call stack entry earlier in the call stack. 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 call stack, transferring important information to a previous program. When messages are sent to a program that ends

without moving its messages up the stack, the programs left in the stack cannot receive those messages unless:

- They use the message key of those messages.
- They use the List Job Log (QMHLJOBL) API.

Assume a program is sent several diagnostic messages and one escape message in response to an error. Another program earlier in the stack can handle the error. If the first program uses the QMHMOVPM API to move all the diagnostic messages to the program earlier in the stack, it then uses "Resend Escape Message (QMHRSNEM) API" on page 40-40 to send the escape message to the calling program.

Required Parameter Group

Message key
INPUT; CHAR(4)
When moving a single, specific message, use the key to that message for this parameter. 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 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 40-2.

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.

When moving a single message by specifying the message key in the message key parameter, use 0 for the number of 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.

Call message queue
INPUT; CHAR(*)
The name of the call message queue to move the messages to, or the name of the call stack entry to start counting from when using a value other than 0 for the call stack counter parameter. The call stack entry you specify must be in the call 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).

Call stack counter
INPUT; BINARY(4)
A number identifying the location in the call stack of the call message queue to which messages are being moved. The number is relative to the name specified in the call message queue parameter. It indicates how many calls up the call stack the target call message queue is from the current one specified in the call message queue parameter. Valid values follow:

0 Move the messages to the message queue of the name specified in the call message queue parameter. You cannot use 0 when the call message queue parameter specifies * to designate the current call message queue.

1 Move the messages to the message queue of the call stack entry that called the program specified in the call message queue parameter.

n (any positive number)
Move the messages to the queue of the nth call stack entry earlier in the stack from the program specified in the call message queue parameter.

You can use any positive number that does not exceed the actual number of call stack entries in the call 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.

Optional Parameter Group

Length of call message queue
INPUT; BINARY(4)
The length of the call message queue name. Valid values for this parameter are any value from 1 through 256. If this parameter is not used, the call message queue name is assumed to be 10 characters in length.

Call stack entry qualification
INPUT; CHAR(20)
The name of the module and the ILE program to further qualify the procedure specified on the call message queue parameter. The first 10 characters specify the

module name, and the second 10 characters specify the ILE program name. If this parameter is not used, only the call message queue parameter is used to identify the call. The following special value may be used:

***NONE** This value is used for the module or ILE program name, or both. When *NONE is specified for one of the names, then that name is not used when searching for the qualified procedure. If *NONE is specified for both names, only the call message queue parameter is used to identify the call stack entry.

If this parameter is not used, only the call message queue parameter is used to identify the call stack entry.

Error Messages

CPF24A3 E Value for call stack counter parameter not valid.
CPF24A5 E Value of &1, for number of message types, not valid.
CPF24BF E Module or ILE program name is blank.
CPF24B3 E Message type &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B5 E Not able to move message.
CPF24B7 E Value &1 for call stack entry name length not valid.
CPF24B9 E When call stack entry name is *, module name and bound program name must be *NONE.
CPF241B E Message type &1 in system program is not valid.
CPF2410 E Message key not found in message queue &1.
CPF247A E Call stack entry &3 contained in module &1 and ILE program &2 not found.
CPF2471 E Length of field not valid.
CPF2479 E Call stack entry &1 not found.
CPF2508 E Messages may not be moved to the same call stack entry as the mover entry.
CPF2509 E Message key &2 refers to a message that is not on the mover's call stack entry.
CPF3CF1 E Error code parameter not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Promote Message (QMHPRMM) API

Parameters

Required Parameter Group:

1	Invocation pointer	Input	Pointer
2	Call stack counter	Input	Binary(4)
3	Message key	Input	Char(4)
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	Message type	Input	Char(10)
9	Message severity	Input	Binary(4)
10	Log option	Input	Char(1)
11	Priority	Input	Char(10)
12	New message key	Output	Char(4)
13	Error code	I/O	Char(*)

The Promote Message (QMHPRMM) API promotes an escape or status message that has been sent to a call stack entry. The API replaces the message with a new escape or status message to the same call stack entry. An escape message may be promoted to another escape message or to a status message. A status message may be promoted to an escape message or to another status message. The stored severity of the new message may be overridden. The new message may be added to the job log, and the initial priority of the new message may be changed.

The message to be promoted must be an active escape message or status message, existing on the message queue of the specified call stack entry. This API causes the existing message to be handled. The source call stack entry of the new message is the same as the source call stack entry of the promoted message. The message key of the new message is returned to the caller of this API. If the new message is not monitored by the target call stack entry, an error may be returned to the caller of this API. The new message key may not be returned.

Authorities and Locks

Message File Authority
*USE
Message File Library Authority
*USE

Required Parameter Group

Invocation pointer
INPUT; POINTER
The invocation pointer to the call stack entry receiving the escape message or status message that is to be promoted. When the call stack counter parameter is other than 0, the invocation pointer points to the call stack entry from which to start counting earlier in the call stack. This locates the call stack entry that received the escape message or status message being promoted. The call stack entry you specify must be in the call

stack. A null invocation pointer may be specified. If this pointer is not set, the call stack entry that called the QMHPRMM API is used.

Call stack counter
INPUT; BINARY(4)
A number identifying the location in the call stack of the call stack entry receiving the escape or status message that is to be promoted. The number is relative to the call stack entry specified in the invocation pointer parameter. It indicates how many calls up the call stack the call stack entry is from the one specified in the invocation pointer parameter. Valid values follow:

0 Promotes the message that was sent to the call stack entry specified in the invocation pointer parameter.
1 Promotes the message sent to the call stack entry that called the call stack entry specified in the invocation pointer parameter.
n (any positive number)
Promotes the message sent to the nth call stack entry earlier in the stack from the call stack entry specified in the invocation pointer parameter.

You can use any positive number that points to an actual call stack entry in the call stack.

Message key
INPUT; CHAR(4)
The message key of the existing message that is to be promoted.

Message identifier
INPUT; CHAR(7)
The identifying code for the new predefined message that replaces the message being promoted.

Qualified message file name
INPUT; CHAR(20)
The name of the message file and the library in which the new predefined message resides. The first 10 characters specify the file name, and the second 10 characters specify the library. You must specify both the message file name and the library name. 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 data to insert in the predefined message's substitution variables.

If there is no data to insert, a value of 0 should be specified for the length of message data parameter.

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 message data, in bytes. Valid values are 0 through 32767.

Message type
INPUT; CHAR(10)
The type of the new message. You must specify one of these values:

***ESCAPE** Escape message
***STATUS** Status message

You may promote an escape message to another escape message or to a status message. You may promote a status message to an escape message or to another status message.

Message severity
INPUT; BINARY(4)
A number identifying the message severity that overrides the severity that is stored in the message description for the message.

Valid values follow:

-1 The message severity is not to be overridden.
0–99 The severity for the new message that replaces the message being promoted. For a list of the severity codes and their meanings, see the *CL Programmer's Guide*.

Log option
INPUT; CHAR(1)
A value that indicates whether the new message is to appear in the job log.

Valid values follow:

0 The new message is not to appear in the job log.
1 The new message is to appear in the job log. However, if the exception monitor for the new message indicates that it is not to be enqueued, it does not appear in the job log. This parameter value is only valid when the message type parameter is *ESCAPE. If the message type parameter is *STATUS, an error is returned to the caller of this API.

Priority
INPUT; CHAR(10)
The initial priority of the new message. You must specify one of these values:

***CONTINUE** Continue. The initial priority of the new message is the same as the current priority of the message that is being promoted.
***LERETRY** Retry. The initial priority of the new message is 135, so that the first condition handler that will get control is the ILE condition manager.

*LEDFT	LE/400 Default. The initial priority of the new message is 225, which indicates LE default action. This causes the exception to be moved up to the next call stack entry where processing continues at the first handler condition.

New message key
OUTPUT; CHAR(4)
The message key of the new message that replaces the promoted message.

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 call stack counter parameter not valid.
CPF24B3 E Message type &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B6 E Length of &1, not valid for message text or data.
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.
CPF2419 E Message identifier &1 not found in message file &2 in &3.
CPF243A E Invocation pointer parameter not valid.
CPF243B E Message severity &1 not valid.
CPF243C E Log option &1 not valid.
CPF243D E Priority &1 not valid.
CPF243E E Message is not an active exception.
CPF243F E Cannot promote message of type other than *ESCAPE or *STATUS.
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.
CPF8100 E Any damage notification.
CPF9830 E Cannot assign library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 time	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 40-34.

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
Message Queue Lock	If a wait time is specified and the API must wait the specified amount of time, an exclusive lock is put on the message queue.

Required Parameter Group

Message information
OUTPUT; CHAR(*)
The variable that receives the information returned, in the format specified by the format name parameter, of the length specified by 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 40-30.
RCVM0200 All message information. For details, see "RCVM0200 Format" on page 40-30.

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 40-29.

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 40-29.

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 and there is no 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, you can use hexadecimal zeros 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 follow:

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 specified time, most fields in the RCVM0100 or RCVM0200 output 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 follow:

***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 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. 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 the sender's 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 to receive the last message on the queue.

Message Type	Message Key	Description
*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 to receive the first message on the queue.
*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 40-31.

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 40-31.

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 name
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 name
120	78	CHAR(4)	Receiving program's instruction number
124	7C	CHAR(1)	Sending type
125	7D	CHAR(1)	Receiving type
126	7E	CHAR(9)	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

RCVM0300 Format

The following table lists the fields returned in the RCVM0300 format. For more information about each item of information, see "Field Descriptions" on page 40-31.

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(9)	Alert option
64	40	CHAR(16)	Reserved
80	50	BINARY(4)	Length of message data or text returned
84	54	BINARY(4)	Length of message data or text available
88	58	BINARY(4)	Length of message available
92	5C	BINARY(4)	Length of message available
96	60	BINARY(4)	Length of message help returned
100	64	BINARY(4)	Length of message help available
104	68	BINARY(4)	Length of sender information returned
108	6C	BINARY(4)	Length of sender information available
112	70	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
		CHAR(*)	Sender information

Sender Information Format: The following table lists the fields for the sender information format of the RCVM0300 format. For more information about each item of information, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Sending job
10	A	CHAR(10)	Sending user profile
20	14	CHAR(6)	Sending job's number
26	1A	CHAR(7)	Date sent
33	21	CHAR(6)	Time sent
39	27	CHAR(1)	Sending type
40	28	CHAR(1)	Receiving type
41	29	CHAR(12)	Sending program name
53	35	CHAR(10)	Sending module name
63	3F	CHAR(256)	Sending procedure name
319	13F	CHAR(1)	Reserved
320	140	BINARY(4)	Number of statement numbers or instruction numbers available for the sending program or procedure
324	144	CHAR(30)	Sending program's statement numbers or instruction numbers
354	162	CHAR(10)	Receiving program name
364	16C	CHAR(10)	Receiving module name
374	176	CHAR(256)	Receiving procedure name
630	276	CHAR(10)	Reserved
640	280	BINARY(4)	Number of statement numbers or instruction numbers available for the receiving program or procedure.
644	284	CHAR(30)	Receiving program's statement number or instruction number
674	2A2	CHAR(46)	Reserved

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 when the message is sent to the QHST message queue.
**NO* No alert is sent.
**UNATTEND* An alert is sent immediately when the system is running in unattended mode (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. Also, it is always 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 following occurs:

- The value of the bytes returned field is 8.
- The value of the bytes available field is 0.
- 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 online help information, in bytes. If an immediate message is received, the value of this field is zero.

Length of message help returned. The length of the online help information, 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.

Length of sender information available. The length, in bytes, of information available in the sender information format.

Length of sender information returned. The length, in bytes, of information returned in the sender information format,

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. It 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 that contains pointer data, each pointer must start on a 16-byte boundary. If you are running at security level 50, the pointer data is invalidated.

The API can truncate the data or text to fit the available space. If the field contains the text of an immediate message and is truncated in the middle of double-byte character set (DBCS) data, the API returns only complete DBCS characters. It 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 name. 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. It 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

Name of the procedure receiving the message. This field is blank if the message was sent by an original program model (OPM) program.

Number of statement numbers or instruction numbers available for the sending program or procedure. For OPM programs and nonoptimized procedures, this count is 1. For optimized procedures, this count can be greater than 1. In this case, each statement number represents a potential point at which the message could have been sent. If the mapping table information is removed from the program, this field returns a count of zero and no statement numbers are available.

Number of statement numbers or instruction numbers available for the receiving program or procedure. For OPM programs and nonoptimized procedures, this count is 1. For optimized procedures, this count can be greater than 1. In this case, each statement number represents a potential point at which the message could have been received. If the mapping table information has been removed from the program, this field returns a count of zero and no statement numbers are available.

Receiving module name. The name of the module that contains the procedure receiving the message. This field is blank if the message was received by an OPM program.

Receiving procedure name. The name of the module that contains the procedure receiving the message. This field is blank if the message was received by an OPM program.

Receiving program name. The name of the program receiving the message, or the Integrated Language Environment (ILE) program name that contains the procedure receiving the message.

Receiving program's statement number or instruction number. This field can contain up to three statement numbers or instruction numbers. When the receiving type is a procedure within an ILE program, this field contains statement numbers. When the receiving type is a program, this field contains an instruction number. Each statement number can be up to 10 characters in length. The instruction number is 4 characters in length. Each statement number or instruction number is left-justified in a 10-character partition of the 30-character field. The first statement number or instruction number is in the leftmost 10 characters and the third in the rightmost 10 characters. The unused parts of this field are set to blanks.

Receiving type. The type of program that received the message. Valid values follow:

0 The message was sent to a program.
1 The message was sent to a procedure within an ILE program.

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 module name. The name of the module that contains the sending message. This field is blank if the message was not sent by a procedure within an ILE program.

Sending procedure name. The name of the procedure sending the message. This field is blank if the message was not sent by a procedure within an ILE program.

Sending program name. The program name or ILE program name that contains the procedure sending the message.

Sending program's statement numbers or instruction numbers. This field can contain up to three statement numbers or an instruction number. When the sending type is a procedure within an ILE program, this field contains statement numbers. When the sending type is a program, this

field contains an instruction number. Each statement number can be up to 10 characters in length. The instruction number is 4 characters in length. Each statement number or instruction number is left-justified in a 10-character partition of the 30-character field. The first statement number or instruction number is in the leftmost 10 characters and the third in the rightmost 10 characters. The unused parts of this field are set to blanks.

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 type. The type of the sender (whether it is a program or procedure). Possible values and their meanings follow:

0 Sender is a program
1 Sender is a procedure within an ILE program.

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.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9830 E Cannot assign library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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	Call message queue	Input	Char(*)
5	Call 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(*)

Optional Parameter Group:

11	Length of call message queue	Input	Binary(4)
12	Call stack entry qualification	Input	Char(20)

The Receive Program Message (QMHRCVPM) API receives a message from a call message queue or external message queue and returns information describing the message.

To receive a message from nonprogram message queues see "Receive Nonprogram Message (QMHRCVM) API" on page 40-27.

Authorities and Locks

Message File Authority *USE
Message File Library Authority *USE

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, RCVM0200, or RCVM0300 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 40-30.
RCVM0200 All message information. For details, see "RCVM0200 Format" on page 40-30.
RCVM0300 All message information. Complete sender information for the message being received. For details, see "RCVM0300 Format" on page 40-30.

Call message queue
INPUT; CHAR(*)
The name of the call message queue from which to receive messages, or the name of the call stack entry to start counting from when using a value other than 0 for the call stack counter parameter. The call stack entry you specify must be in the call stack. You can specify a call stack entry by name or use one of these special values:

***** The message queue of the current call stack entry (that is, the program receiving the message).
***EXT** The external message queue. The call stack counter parameter is ignored. You cannot receive escape messages from this queue.

Call stack counter
INPUT; BINARY(4)
A number identifying the location in the call stack of the call message queue from which messages are received. The number is relative to the call stack entry specified in the call message queue parameter. It indicates how many calls up the call stack the originating call message queue is from the one specified in the call message queue parameter. Valid values and their meanings are:

0 Receive the message from the message queue of the call stack entry specified in the call message queue parameter.
1 Receive the message from the message queue of the caller of the call stack entry specified in the call message queue parameter.
n (any positive number)
Receive the message from the message queue of the nth call stack entry up the stack from the call stack entry specified in the call message queue parameter.

You can use any positive number that does not exceed the actual number of call stack entries in the call 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 40-36.

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 40-36.

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 and there is no 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 *NEXT, you can use *TOP for the message key to receive the next message after the last request received.

When the message type is *NEXT or *PRV, you can use hexadecimal zeros 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 follow:

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 specified time, most fields in the RCVM0100, RCVM0200 or RCVM0300 output 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 follow:

***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.

Optional Parameter Group

Length of call message queue
INPUT; BINARY(4)
The length of the call message queue parameter. Valid values for this parameter are any value greater than or equal to 1 and less than or equal to 256. If the value is not valid, an error occurs. If this parameter is not used, the call message queue is assumed to be 10 characters in length.

Call stack entry qualification
INPUT; CHAR(20)
This parameter is used to further qualify the procedure identified in the call message queue parameter. Specify the names of the module and the ILE service program that contains the procedure. The first 10 characters specify the module name, and the second 10 characters specify the ILE program name. The following special value may be used:

**NONE* This can be used for the module name or the ILE program name, or both. When *NONE is specified for one of the names, then that name is not used when searching for the qualified procedure. If *NONE is specified for both names, only the call message queue parameter is used to identify the call stack entry.

If this parameter is not used, only the call message queue parameter is used to identify the call stack entry.

Message Types and Message Keys

For general information about message types and message keys, see "Message Types and Message Keys" on page 40-29.

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	If message key is blanks, then this receives a message of any type except a sender's copy or request. If the message key is not blank, this receives the requested message. If the message is a sender's copy type message, the associated reply is received.
*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.

Message Type	Message Key	Description
*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. If *TOP or hexadecimal zeros are specified, the search for the next message begins after the last request message is received. If no request messages have been received, the search starts at the top of the message queue.
*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 call stack counter parameter 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.
CPF24BF E Module or ILE program name is blank.
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.
CPF24B7 E Value &1 for call stack entry name length not valid.
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.
CPF2423 E Variable specified in Sender parameter less than 80 bytes.
CPF2449 E Message that should be a reply, is not a reply.
CPF247A E Call stack entry &3 contained in module &1 and ILE program &2 not found.
CPF2479 E Call stack entry &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.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9830 E Cannot assign library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 or a group of messages from a message queue, 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" on page 40-38.

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 follow:

***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.
CPF3C36 E Number of parameters, &1, entered for this API not valid.
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.
CPF9838 E User profile storage limit exceeded.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Remove Program Messages (QMHRMVPM) API

Parameters

Required Parameter Group:

1	Call message queue	Input	Char(*)
2	Call stack counter	Input	Binary(4)
3	Message key	Input	Char(4)
4	Messages to remove	Input	Char(10)
5	Error code	I/O	Char(*)

Optional Parameter Group:

6	Length of call message queue	Input	Binary(4)
7	Call stack entry qualification	Input	Char(20)
8	Remove unhandled exceptions	Input	Char(10)

The Remove Program Messages (QMHRMVPM) API removes messages from call 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 40-37.

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 must 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

Call message queue
INPUT; CHAR(*)
The name of the call message queue from which to remove messages, or the name of the call stack entry to start counting from when using the call stack counter parameter. The call stack entry you specify must be in the call stack. You can specify a call stack entry by name or use one of these special values:

* The message queue of the current call stack entry (that is, the call stack entry removing the messages).

***ALLINACT**
All message queues for inactive call stack entries. The messages to remove parameter must specify *ALL. The call stack counter parameter is ignored.

***EXT** The external message queue. The call stack counter parameter is ignored.

If you specify a message key and the messages to remove parameter specifies *BYKEY, this parameter is ignored.

Call stack counter
INPUT; BINARY(4)
A number identifying the location in the call stack of the call message queue from which to remove messages. The number is relative to the call stack entry specified in the call message queue parameter. It indicates how many calls up the call stack the targeted call message queue is from the one specified in the call message queue parameter. Valid values follow:

0 Remove the messages from the queue of the call stack entry specified in the call message queue parameter.

1 Remove the messages from the queue of the call stack entry that called the call stack entry specified in the call message queue parameter.

n (any positive number)
Remove the messages from the queue of the nth call stack entry up the stack from the call stack entry specified in the call message queue parameter.

You can use any positive number that does not exceed the actual number of call stack entries in the call 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 call stack entries (*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 call message queue and call 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 follow:

***ALL** All messages in the call message queue.
***BYKEY** The single message specified in the message key parameter.
***KEEPRQS**
All messages in the call message queue except requests.
***NEW** All new messages in the call message queue. New messages are those that have not been received.
***OLD** All old messages in the call 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.

Optional Parameter Group

Length of call message queue
INPUT; BINARY(4)
The length of the call message queue parameter. Valid values for this parameter are greater than or equal to 1, and less than or equal to 256. If the value is not valid, an error occurs. If this parameter is not used, the call message queue is assumed to be 10 characters in length.

Call stack entry qualification
INPUT; CHAR(20)
To further qualify the procedure identified in the call

message queue parameter, specify the names of the module and the ILE program that contains the procedure. The first 10 characters specify the module name, and the second 10 characters specify the ILE program name. The following special value may be used:

**NONE* This can be used for the module name or the ILE program name, or both. When *NONE is specified for one of the names, then that name is not used when searching for the qualified procedure. If *NONE is specified for both names, only the call message queue parameter is used to identify the call stack entry.

If this parameter is not used, only the call message queue parameter is used to identify the call stack entry.

Remove unhandled exceptions
INPUT; CHAR(10)
Whether to remove unhandled exceptions in the identified call message queue. The exception handling support for some languages allows exceptions to not be handled. Valid values follow:

**YES* Indicates to remove any unhandled exceptions in the identified call message queue. This is the default when this parameter is not used.
**NO* Indicates to not remove any unhandled exceptions in the identified call message queue.

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 call stack counter parameter not valid.
CPF24A6 E Value for messages to remove not valid.
CPF24BF E Module or bound-program name is blank.
CPF24B4 E Severe error while addressing parameter list.
CPF24B7 E Value &1 for call stack entry name length not valid.
CPF24B8 E Value &1 for remove unhandled exceptions not valid.
CPF241A E Clear option &1 in system program is not valid.
CPF2410 E Message key not found in message queue &1.
CPF247A E Call stack entry &3 contained in module &1 and bound program &2 not found.
CPF2479 E Call stack entry &1 not found.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 call stack entry to the previous call stack entry in the call stack. 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 exception message handling. If a call stack entry 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. You can use the QMHRSNEM API to forward the escape message to the previous call stack entry in the call stack. The call stack entry 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 40-23.

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

CPF24BC E No escape message to resend.
CPF24B4 E Severe error while addressing parameter list.
CPF2410 E Message key not found in message queue &1.
CPF2524 E Exception handler not available because of reason code &1.
CPF2550 E Exception message sent to a deleted program or procedure.
CPF3CF1 E Error code parameter not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 description of a predefined message. The **message description** 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 Non-program 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 from the QMHRCVM or the QMHRCVPM API 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" on page 40-42.
RTVM0200 All message information. For details, see "RTVM0200 Format" on page 40-42.

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 32767.

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."

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
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			
Dec	**Hex**	**Type**	**Field**
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 follow:

**DEFER* An alert is sent after local problem analysis.

**IMMED* An alert is sent immediately when the message is sent to the QHST message log.

**NO* No alert is sent.

**UNATTEND* An alert is sent immediately when the system is running in unattended mode (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 APIs 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 40-31.

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 follow:

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 &1, not valid for message text or data.
| 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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Nonprogram Message Queue Attributes (QMHRMQAT) API

Parameters

Required Parameter Group:

1	Message queue information	Output	Char(*)
2	Length of message queue information	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified message queue name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Nonprogram Message Queue Attributes (QMHRMQAT) API provides information about the attributes of a nonprogram message queue.

Authorities and Locks

Message queue *USE
Message queue library *READ

Required Parameter Group

Message queue information
OUTPUT; CHAR(*)
The parameter to receive the message queue information. The format of the returned data is specified in "RMQA0100 Format" on page 40-44.

Length of message queue information
INPUT; BINARY(4)
The length of the message queue information parameter described in "RMQA0100 Format" on page 40-44. If the length specified is larger than the actual size of the message queue information parameter, the results may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the information to be returned. You must use the following format name:

RMQA0100 This format is described in "RMQA0100 Format."

Qualified message queue name
INPUT; CHAR(20)
The message queue and the library in which the message queue is located. The first 10 characters contain the message queue name, and the second 10 characters contain the message queue library. 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.

RMQA0100 Format

The following table shows the information returned in the message queue information parameter for the RMQA0100 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(10)	Message queue used
18	12	CHAR(10)	Message queue library used
28	1C	BINARY(4)	Number of messages on queue
32	20	BINARY(4)	Current storage size
36	24	BINARY(4)	Increment storage size
40	28	BINARY(4)	Number of increments
44	2C	BINARY(4)	Maximum increments
48	30	BINARY(4)	Severity code filter
52	34	CHAR(7)	Delivery
59	3B	CHAR(10)	Break-handling program name
69	45	CHAR(10)	Break-handling program library
79	4F	CHAR(4)	Force to auxiliary storage
83	53	CHAR(50)	Text description

Field Descriptions

Break-handling program name. The name of the program called when messages are sent to this message queue. The delivery field must be set to *BREAK. The severity of the message sent to the queue must be equal to or greater than the value shown in the severity code filter field.: The field is set to *DSPMSG if the queue is in break mode and there is no break-handling program set.

The field is set to blanks if the queue is not in break mode.

Break-handling program library. The library specified for the break-handling program.

If the message queue is not in break mode, or the break-handling program name is specified as *DSPMSG, this field is set to blanks.

The following special values or a specific library may be returned:

**LIBL* The library list is used to locate the program.
**CURLIB* The current library for the job is used to locate the program.

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. The number of bytes returned is always less than or equal to the bytes available field or the length of message queue information parameter.

Current storage size. The current storage size of the message queue in bytes.

Delivery. Specifies how the messages that are sent to this message queue are delivered. The method of delivery is in effect only as long as the message queue is allocated to a job. When the queue is no longer allocated, the delivery mode is changed to *HOLD for work station, system operator, and user message queues.

The possible values follow:

**HOLD* The messages are held in the message queue until they are requested by a user or program.
**BREAK* When a message arrives at the message queue, the job to which the message queue is allocated is interrupted. Then the program specified on the break-handling program name field is called. If the break-handling program name is set to *DSPMSG, the Display Message (DSPMSG) command is processed.

| *NOTIFY When a message arrives at the message queue, an interactive job to which the message queue is allocated is notified. The message light turns on and a buzzer sounds (if the feature is available).
| *DFT Messages requiring replies are answered with their default reply. No messages are added to the message queue unless the message queue is QSYSOPR.

| **Force to auxiliary storage**. Whether changes made to the message queue description or messages added to or removed from the queue are immediately forced into auxiliary storage. If the value is *YES and a system failure occurs, the changes to the message queue are not lost.

| One of the following modes are returned:

| *YES All changes to the message queue description and to the messages in the queue are immediately forced into auxiliary storage.
| *NO Changes made to the message queue description, including its messages, are not immediately forced into auxiliary storage.

| **Increment storage size**. The number of bytes added each time the message queue is increased in size.

| **Maximum increments**. The maximum number of times the message queue can be increased in size.

| **Message queue library used**. The actual library name that contains the message queue.

| **Message queue used**. The name of the message queue whose attributes were returned.

| **Number of increments**. The number of times the message queue has been increased in size.

| **Number of messages on queue**. The number of messages currently on the message queue.

| **Severity code filter**. The lowest severity level that a message can have and still be delivered to a user in break or notify mode. Messages arriving at the message queue whose severities are lower than that specified here do not interrupt the job or turn on the message waiting light. Valid values are 0 through 99.

| **Text description**. Text that briefly describes the message queue.

| Error Messages

| 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.
| CPF2477 E Message queue &1 currently in use.
| CPF2536 E Value &1, for the length of message queue information not valid.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF8176 E Message queue for device description &4 damaged.
| CPF9830 E Cannot assign library &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

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/0	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, no error is returned. (For example, there are no request messages prior to or after the one identified by the message key passed in.) Instead, bytes available is set to 0 to indicate no message was found.

Length of message information
INPUT; BINARY(4)
The size of the area to contain the message information, in bytes. The minimum size is 8. If this value is larger than the actual size of the storage allocated for the message information parameter, the result may not be predictable.

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 follow:

***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 need not refer to a message on the job message queue. The search begins with the message on the queue that has the key closest to this value. 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		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(4)	Message key
12	C	CHAR(20)	Reserved
32	20	BINARY(4)	Length of request message text returned
36	24	BINARY(4)	Length of request message text available
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		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(4)	Message key
12	C	CHAR(10)	Program name or ILE service program name
22	16	CHAR(1)	Call stack entry type
23	17	CHAR(10)	Module name
33	21	CHAR(256)	Procedure name
289	121	CHAR(19)	Reserved
308	134	BINARY(4)	Length of request message text returned
312	138	BINARY(4)	Length of request message text available
316	13C	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 the data, this value is less than bytes available. When no request messages could be retrieved from the job message queue, the following occurs:

- The value of the bytes returned field is set to 8.
- The value of the bytes available field is set to 0.
- The remaining fields are unchanged.

Call stack entry type. The type of call stack entry receiving the message. The possible values follow:

0 The receiving call stack entry is a program.

1 The receiving call stack entry is a procedure within an ILE program.

Length of request message text available. The length of all available request message text, in bytes.

Length of request message text returned. The actual length of data in the request message text field. If the returned length is equal to the available length, all request message text was returned. If the message information parameter is not sufficiently large to hold all the text, the returned length is less than the available length and the text is truncated.

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. This retrieves 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.

Module name. The name of the module that contains the ILE procedure receiving the message. If the message was sent to an OPM program, this field is blank.

Procedure name. The name of the ILE procedure receiving the message. If the message was sent to an OPM program, this field is blank.

Program name or ILE service program name. The name of the program receiving the request message, or the name of the ILE program that contains the procedure receiving the message.

Request message text. The text of the request message. If you are retrieving CL commands, the maximum length of a CL command is 6000 bytes.

Reserved. An ignored field.

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.
CPF2415 E End of requests.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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. This 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 to your program message queue. It 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. This message is returned 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 6000.

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. Use 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 call stack entry's call 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 &1, not valid for message text or data.
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.
CPF2429	D	Message to work station &1 did not break.
CPF2469	E	Error occurred when sending message &1.
CPF2401	E	Not authorized to library &1.
CPF2403	E	Message queue &1 in &2 not found.
CPF2408	E	Not authorized to message queue &1.
CPF2421	E	Message not sent. &1 and &2 not work station message queue.
CPF2460	E	Message queue &1 could not be extended.
CPF2467	E	&3 message queue &1 in library &2 logically damaged.
CPF2477	E	Message queue &1 currently in use.
CPF2481	E	Work station message queue not available.
CPF8100	E	All CPF81xx messages could be returned. xx is from 01 to FF.
CPF3CF1	E	Error code parameter not valid.
CPF9830	E	Cannot assign library &1.
CPF9872	E	Program &1 in library &2 ended. Reason code &3.

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 call message queue or the external message queue, see "Send Program Message (QMHSNDPM) API" on page 40-51.

Before coding your call to the QMHSNDM API, see "Dependencies among Parameters" on page 40-50.

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 to your call message queue. The API 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. This message is sent 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.

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 0–32767
Immediate text 1–6000

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. 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 40-27, 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 40-55.

For descriptions of the message types, see "Message Types" on page 40-2. For details about coding the other parameters when sending a particular type of message, see "Dependencies among Parameters" on page 40-50.

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 call stack entry's call 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	0-32767	1-6000	0-32767	1-6000
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

Parameter	Message Type			
	*COMP, *DIAG, *INFO		*INQ	
	Predefined	Immediate	Predefined	Immediate
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 &1, not valid for message text or data.
| CPF2428 E Only one message queue allowed for *INQ and *NOTIFY type messages.
| CPF2433 E Function not allowed for system log message queue &1.
| CPF2435 E System reply list not found.
| CPF2469 E Error occurred when sending message&1.
| 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.
| CPF2411 E Not authorized to message file &1 in &2.
| CPF2421 E Message not sent. &1 and &2 not work station message queue.
| CPF2460 E Message queue &1 could not be extended.
| CPF2467 E &3 message queue &1 in library &2 logically damaged.
| CPF2477 E Message queue &1 currently in use.
| CPF2548 E Damage to message file &1 in &2.
| CPF2557 E System reply list damaged.
| CPF2558 E System reply list currently in use.
| CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
| 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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Send Program Message (QMHSNDPM) API

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	Call message queue	Input	Char(*)
7	Call stack counter	Input	Binary(4)
8	Message key	Output	Char(4)
9	Error code	I/O	Char(*)

Optional Parameter Group:

10	Length of call message queue	Input	Binary(4)
11	Call stack entry qualification	Input	Char(20)
12	Display program messages screen wait time	Input	Binary(4)

The Send Program Message (QMHSNDPM) API sends a message to a call 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 call stack entry.) This API allows the current call stack entry 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 40-48.

Before coding your call to the QMHSNDPM API, see "Dependencies among Parameters" on page 40-53.

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 0–32767
Immediate text 1–6000

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 40-55.

For descriptions of the message types, see "Message Types" on page 40-2. For details about coding the other parameters when sending a particular type of message, see "Dependencies among Parameters" on page 40-53. For further information about working with each type, see "Additional Information about Message Types" on page 40-53.

Call message queue
INPUT; CHAR(*)
The name of the call message queue to send the messages to, or the name of the call stack entry to start counting from when using a value other than 0 for the call stack counter parameter. The call stack entry you specify must be in the call stack. You can specify a call stack entry by name or use one of these special values:

***** The message queue of the current call stack entry (that is, the call stack entry sending the message).
***EXT** The external message queue. The call stack counter parameter is ignored. You cannot send escape messages to this message queue. If you are sending an inquiry message, you must specify this message queue.

Call stack counter
INPUT; BINARY(4)
A number identifying the location in the call stack of the call message queue receiving the messages. The number is relative to the name specified in the call message queue parameter. It indicates how many calls up the call stack this call message queue is from the one specified in the call message queue parameter.

Valid values follow:

0 Send the message to the queue of the call stack entry specified in the call message queue parameter.
1 Send the message to the queue of the caller of the call stack entry specified in the call message queue parameter.
n (any positive number)
Send the message to the queue of the nth caller up the stack from the call stack entry specified in the call message queue parameter.

You can use any positive number that does not exceed the actual number of call stack entries in the call 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 status. For a 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.

Optional Parameter Group

Length of call message queue
INPUT; BINARY(4)
The length of the call message queue parameter. Valid values for this parameter are any value greater than or equal to 1 and less than or equal to 256. If the value is not valid, an error occurs. If this parameter is not used, the call message queue is assumed to be 10 characters in length.

Call stack entry qualification
INPUT; CHAR(20)
To further qualify the procedure identified in the call message queue parameter, specify the names of the module and ILE program that contain the procedure. The first 10 characters specify the module name, and the second 10 characters specify the ILE program name. The following special value may be used:

**NONE* This can be used for the module name or the ILE program name, or both. When *NONE is specified for one of the names, that name is not used when searching for the qualified procedure. If *NONE is specified for both names, only the call message queue parameter is used to identify the call stack entry.

If this parameter is not used, only the call message queue parameter is used to identify the call stack entry.

Display program messages screen wait time
INPUT; BINARY(4)
The amount of time in seconds to wait on the display program messages screen for any user action when a message is sent to an external message queue. If there is no user action within the specified time, the control returns to the statement following the call of this API. Valid values follow:

–1 This is the default. If this parameter is not used, this value is assumed. This value specifies that there is no maximum time to wait. The wait is 24 hours before control is returned.
0 Send the message, but do not show the display program messages screen.
>0 The amount of time to wait in seconds for user action. If there is no user action within the time, then control returns. If the display file currently being displayed is defined with the restore display attribute of *NO, the Display Program Messages screen wait time parameter is ignored. A value of -1 is assumed.

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.

Parameter	Message Type						
	*COMP, *DIAG, *INFO		*INQ		*STATUS	*ESCAPE, *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	0-32767	1-6000	0-32767	1-6000	0-32767	0-32767	1-6000
Call message queue	Required	Required	*EXT	*EXT	Required	Required	Required
Call stack counter	0-n	0-n	Ignored	Ignored	0-n	0-n	0-n
Message key	Returned	Returned	Returned	Returned	Not used	Returned	Returned
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 call message queues or to the external message queue.

Escape Messages: You can send escape messages only with a message identifier and only to a call message queue. You cannot send them to the external message queue.

An escape message sent to a previous caller's call message queue ends the current call stack entry. Sending the escape message causes all the call stack entries up to but not including the call stack entry receiving the escape message to end. For example, if three call stack entries are active and the third call stack entry sends an escape message to the first, then the second and third call stack entries are ended.

Informational Messages: You can send informational messages as immediate messages or with message identifiers to call 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 call message queues. The sending call stack entry would have to end before the receiving call stack entry could receive the message and send a reply. At that point, the sending call stack entry 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 call stack entry's call message queue. Sending an inquiry message lets your call stack entry 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. Call stack entries cannot reply to inquiry messages sent to the external message queue. In batch mode, the default reply is automatically sent as the 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 call message queue or to the external message queue.

The reply message queue is always the sending call stack entry's call 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 call message queue, the action taken depends on two things:

- Whether the receiving call stack entry monitors for the message.
- Whether the receiving call stack entry has specified an external exception handler to handle the message.

When a notify message is sent to the external message queue, the action taken is the same as if an inquiry message were sent, with this exception. Notify messages are not responded to by the system reply list.

Figure 40-1. NOTIFY Message Results

Receiving Call Stack Entry	Sending Call Stack Entry	Default Reply	Action Taken after Notify Message Is Sent
Monitor defined with external handler	Does not end	Not sent	External handler is called
Monitor defined and no external handler	Ends	Not sent	Control is returned to receiving call stack entry
No monitor defined	Does not end	Sent	Control is returned to sending call stack entry

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.

Request Messages: You can send a request message only as an immediate message. You can send it to a call 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 on the external message queue; it does not work with request messages on other queues.

Status Messages: You must send status messages with a message identifier. You can send them to a call 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. This shows the status of running programs. You can use this to reassure users that long programs are still running.

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. CPF9898 is in message file QCPFMSG in library QSYS. This message consists only 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 call stack counter parameter not valid.
CPF24BF E Module or ILE program name is blank.
CPF24B3 E Message type &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF24B6 E Length of &1 not valid for message text or data.
CPF24B7 E Value &1 for call stack entry name length not valid.
CPF24B9 E When call stack entry name is *, module name and ILE program name must be *NONE.
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.
CPF246A E Destination call stack entry not valid.
CPF2467 E &3 message queue &1 in library &2 logically damaged.
CPF2469 E Error occurred when sending message &1.
CPF247A E Call stack entry &3 contained in module &1 and ILE program &2 not found.
CPF2479 E Call stack entry &1 not found.
CPF2489 E Message identifier required for *NOTIFY, *ESCAPE, or *STATUS message.
CPF2493 E Message identifier not allowed with message type *RQS.
CPF2499 E Message identifier &1 not allowed.
CPF2524 E Exception handler not available because of reason code &1.
CPF2531 E Message file &1 in &2 damaged for &3.
CPF2547 E Damage to message file QCPFMSG.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 application 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 any of the following:
  - Receive Message (RCVMSG) command
  - Receive Program Message (QMHRCVPM) API

- Receive Nonprogram Message (QMHRCVM) API
- List Messages (QMHLSTM) API
- List Job Log (QMHLJOBL) 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 follow:

***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 &1, not valid for message text or data.
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.
CPF2432 E Cannot send reply to message type other than *INQ or *NOTIFY.
CPF2433 E Function not allowed for system log message queue &1.
CPF2439 E Reply value entered is not valid.
CPF2440 E Reply must be in range of &1 to &2.
CPF2442 E Reply does not meet specified relations.
CPF2460 E Message queue &1 could not be extended.
CPF2466 E Reply length greater than &1.
CPF2477 E Message queue &1 currently in use.
CPF2547 E Damage to message file QCPFMSG.
CPF2548 E Damage to message file &1 in &2.
CPF3CF1 E Error code parameter not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 12. National Language Support APIs

Chapter 41. National Language Support APIs

This chapter describes the syntax and parameters for the national language support APIs. These APIs consist of the program calls shown in the following list:

Convert Sort Sequence Table (QLGCNVSS) converts a sort sequence table from one CCSID to another.
Retrieve Language ID (QLGRTVLI) retrieves a list of language identifiers.
Retrieve Sort Sequence Table (QLGRTVSS) retrieves a specified sort sequence table.
Scan String for Mixed Data (QLGSCNMX) tests a mixed input string for double-byte characters.
Sort (QLGSORT) provides a generalized sort function.
Sort Input/Output (QLGSRTIO) provides a set of records to be sorted or returns a set of records that have already been sorted.
Truncate Character Data (QLGTRDTA) truncates a CCSID-tagged string of character data to a specified length.
Validate Language ID (QLGVLID) ensures that a language identifier is supported.

The APIs are presented in alphabetical order.

Convert Sort Sequence Table (QLGCNVSS) API

Parameters

Required Parameter Group:

1	Converted sort sequence table	Output	Char(256)
2	Type of returned sort sequence table	Output	Char(1)
3	Substitution values encountered	Output	Char(1)
4	Source sort sequence table	Input	Char(256)
5	CCSID of source table	Input	Binary(4)
6	CCSID of converted table	Input	Binary(4)
7	Error code	I/O	Char(*)

The Convert Sort Sequence Table (QLGCNVSS) API converts a 256-byte sort sequence table from one coded character set identifier (CCSID) encoding to another.

Required Parameter Group

Converted sort sequence table
OUTPUT; CHAR(256)
The converted sort sequence table.

Type of returned sort sequence table
OUTPUT; CHAR(1)
The type of sort sequence table returned. This type testing will be done on every call to the API. Valid values are:

1 A shared-weighted table.
2 A unique-weighted table.

Substitution values encountered
OUTPUT; CHAR(1)
Whether substitution values were involved during the conversion from the source CCSID of the table to the requested CCSID. Valid values are:

0 No substitutions were involved.
1 Substitutions were involved.

Source sort sequence table
INPUT; CHAR(256)
The source sort sequence table to be converted.

CCSID of source table
INPUT; BINARY(4)
The CCSID of the source table. The valid range is 1 to 65533 and 65535.

CCSID of converted table
INPUT; BINARY(4)
The CCSID to which the source table is to be converted. The valid range is 1 to 65533 and 65535.

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 occurred while addressing parameter list.
CPF3BC7 E CCSID &1 outside of valid range.
CPF3BC8 E Conversion from CCSID &1 to CCSID &2 is not supported.
CPF3BC9 E Conversion from CCSID &1 to CCSID &2 is not defined.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Language ID (QLGRTVLI) 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 Language Identifier (QLGRTVLI) API retrieves a list of language identifiers and their descriptions.

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, provided 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.

Format name
INPUT; CHAR(8)
The content and format of the information returned. The possible format names are:

RTVL0100 Basic language identifier format

See "RTVL0100 Format" for a description of this 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.

RTVL0100 Format

Following is the format of the information returned. For a description of the fields in this format, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	Binary(4)	Bytes available
4	4	Binary(4)	Bytes returned
8	8	Binary(4)	Number of language identifiers retrieved
12	B	Binary(4)	CCSID value of descriptive text
16	10	Binary(4)	Offset to start of language identifier array
20	14	Char(*)	Reserved
Note: Format of language identifier array. The following fields are repeated for each language identifier entry.			
0	0	Char(3)	Language Identifier
3	3	Char(40)	Descriptive text

Field Descriptions

Bytes available. The number of bytes of information available.

Bytes returned. The number of bytes of information returned.

CCSID value of descriptive text. The coded character set identifier (CCSID) that all the descriptive texts are encoded in.

Descriptive text. The 40-character descriptive text associated with the language identifier.

Language identifier. The 3-character language identifier.

Number of language identifiers retrieved. The number of language identifiers in the list.

Offset to start of language identifier array. The offset to the start of the language identifier array.

Reserved. An ignored field.

Error Messages

CPF24B4 E Severe error occurred while addressing parameter list.
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.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Sort Sequence Table (QLGRTVSS) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Qualified table name	Input	Char(20)
4	Language identifier requested	Input	Char(10)
5	CCSID of returned table	Input	Binary(4)
6	Format name	Input	Char(8)
7	Error code	I/O	Char(*)

The Retrieve Sort Sequence Table (QLGRTVSS) API returns a 256-byte sort sequence table based on the required input parameters. It will also return some associated job information.

Authorities and Locks

Sort Sequence Table Authority	*USE
Sort Sequence Table Library Authority	*USE
Sort Sequence Table Lock	*SHRNUP

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, provided 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.

Qualified table name
INPUT; CHAR(20)
The sort sequence table to retrieve. The first 10 characters contain the table name, and the second 10 characters contain the name of the library in which the table resides.

The following special values are supported for the table name. These values must be in uppercase, left-justified, and padded with blanks to the right of the value.

***JOB** The sort sequence of the job.
***JOBRUN** The API treats this value the same as *JOB.
***LANGIDUNQ** The unique-weighted sort sequence table that is associated with the language identifier requested parameter.
***LANGIDSHR** The shared-weighted sort sequence table that is associated with the language identifier requested parameter.
***HEX** The sort sequence according to the hexadecimal value of the characters.

Note: Both *JOB and *JOBRUN are accepted to accommodate calls from other programs that accept both values and for which the two have different semantics.

The following special values are supported for the library name. These values must be in uppercase, left-justified, and padded with blanks to the right of the value.

***LIBL** The user's library list.
***CURLIB** The current library.

If a special value is used for the table name, the second 10 characters used for the library name must be blank.

Language identifier requested
INPUT; CHAR(10)
The language identifier of the sort sequence table to be used. All values must be in uppercase and must be padded with blanks to the right of the value. This value will be ignored if the value of the table name parameter is a table object or *HEX. Valid values are:

***JOB** Use the language identifier of the job.
***JOBRUN** The API treats this value the same as *JOB.
Specific language identifier
A valid 3-character language identifier. For example, Danish would be "DAN." See the *National Language Support Planning Guide* for a complete list of valid language identifiers.

Note: Both *JOB and *JOBRUN are accepted to accommodate calls from other programs that accept both values and for which the two have different semantics.

CCSID of returned table
INPUT; BINARY(4)
The coded character set identifier (CCSID) in which the sort sequence table should be returned. The valid values for this parameter are:

0 The CCSID of the job will be used to determine the CCSID of the sort sequence table to be returned.
1–65533 A valid CCSID in this range.
65535 The CCSID of the sort sequence table that is found should be used. No CCSID conversion should be done on the found table.

Format name
INPUT; CHAR(8)
The content and format of the information returned for each table. The format name is:

RSST0100 Basic sort sequence format

See "RSST0100 Format" for a description of this 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.

RSST0100 Format

Following is the format of the information returned. For a description of the fields in this format, see "Field Descriptions" on page 41-4.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	BINARY(4)	Job CCSID value

Offset			
Dec	Hex	Type	Field
12	C	BINARY(4)	CCSID of returned table
16	10	CHAR(1)	Substitution values encountered
17	11	CHAR(1)	Weighting of returned sort sequence table
18	12	CHAR(10)	Returned table name
28	1C	CHAR(10)	Returned table library name
38	26	CHAR(10)	Job sort sequence table name
48	30	CHAR(10)	Job sort sequence table library name
58	3A	CHAR(3)	Job language identifier
61	3D	CHAR(2)	Job country identifier
63	3F	CHAR(256)	Returned sort sequence table

Field Descriptions

Bytes available. The number of bytes of information available.

Bytes returned. The number of bytes of information returned.

CCSID of returned table. The coded character set identifier (CCSID) in which the returned sort sequence table is encoded.

Notes:

1. No conversion is performed if the CCSID tag of the stored table is 65535 or if the job CCSID value is used and it is 65535.
2. All tables created before Version 2 Release 3 will always be assumed to be tagged with a CCSID value of 65535.
3. If *HEX is specified for the sort sequence to get, this value is set to 65535.

Job CCSID value. The CCSID value of this job.

Job country identifier. The country identifier of this job.

Job language identifier. The language identifier of this job

Job sort sequence table library name. The sort sequence table library name of this job.

Job sort sequence table name. The sort sequence table name of this job. It can have one of the special values:

**LANGIDUNQ*
: The unique-weighted sort sequence table that is associated with the language identifier requested parameter.

**LANGIDSHR*
: The shared-weighted sort sequence table that is associated with the language identifier requested parameter.

**HEX*
: The sort sequence according to the hexadecimal value of the characters.

Returned table name. The name of the sort sequence table that was returned.

The value of the returned table name is "*N " if the table returned is the result of a requested sort sequence of *HEX or the result of converting an existing table from its stored CCSID representation to the CCSID requested. (A new table object will not be created in this case.)

Returned table library name. The name of the sort sequence table library that was returned.

The value of the returned table library name is "*N " if the table returned is the result of a requested sort sequence of *HEX or the result of converting an existing table from its stored CCSID representation to the CCSID requested.

If *LIBL or *CURLIB was specified for the qualified table name, the name of the library in which the sort sequence resides will be returned.

Returned sort sequence table. The 256-byte sort sequence table described by the input values.

Substitution values encountered. Whether substitution values were involved during the conversion from the source CCSID of the table to the requested CCSID. A substitution occurs when a character in the source CCSID does not exist in the requested CCSID. The valid values are:

0 No substitutions were involved.
1 Substitutions were involved.

Weighting of returned sort sequence table. The weighting of sort sequence table returned. The valid values are:

1 A shared-weighted table.
2 A unique-weighted table.

Note: Any table created prior to Version 2 Release 3 is always returned as a shared-weighted table.

Error Messages

CPF2115 E Object &1 &2 type *&3 damaged.
CPF24B4 E Severe error occurred while addressing parameter list.
CPF3BC6 E Sort sequence &1 not valid.
CPF3BC7 E CCSID &1 outside of valid range.
CPF3BC8 E Conversion from CCSID &1 to CCSID &2 is not supported.
CPF3BC9 E Conversion from CCSID &1 to CCSID &2 is not defined.
CPF3BCC E Language identifier &1 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.
| CPF3CF1 E Error code not valid.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Not able to allocate &1 in &2.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Scan String for Mixed Data (QLGSCNMX) API

Parameters

Required Parameter Group:

1	Double-byte indicator	Output	Char(1)
2	Input data buffer	Input	Char(*)
3	Length of input data buffer	Input	Binary(4)
4	Error code	I/O	Char(*)

The Scan String for Mixed Data (QLGSCNMX) API tests a mixed EBCDIC input string to determine if the data contains any double-byte characters. If so, the double-byte indicator will be set to 1. It will test a mixed EBCDIC input string that is properly formed. Properly formed means that the string must contain matched pairs of the shift-out (X'0E') and shift-in (X'0F') control characters, if the control characters are present.

Required Parameter Group

Double-byte indicator
OUTPUT; CHAR(1)
Information about the input data. The valid values are:

0 Input data does not contain double-byte data.
1 Input data contains double-byte data.

Input data buffer
INPUT; CHAR(*)
The data to be scanned.

Length of input data buffer
INPUT; BINARY(4)
The length of the input data buffer. The valid range is 1 to 32767 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

| CPF24B4 E Severe error occurred while addressing parameter list.
| CPF2647 E Buffer length is not valid.
| CPF3CF1 E Error code parameter not valid.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Sort (QLGSORT) API

Parameters

Required Parameter Group:

1	Request control block	Input	Char(*)
2	Input data buffer	Input	Char(*)
3	Output data buffer	Output	Char(*)
4	Length of output data buffer	Input	Binary(4)
5	Length of returned data	Output	Binary(4)
6	Error code	I/O	Char(*)

The Sort (QLGSORT) API provides a generalized sort function that can be directly called by any application program. This API can be used to sort data in files or data in an input buffer with a single call. This API can also be used to initialize a sort function where a series of calls using the Sort Input/Output (QLGSRTIO) API is done to repeatedly add data to be sorted as a set of records and return sorted data as a set of records.

The following types of sort operations can be done with the API:

- Sort up to 32 input files and produce 1 to 32 output files, each containing the full list of sorted records.
- Sort up to 32 input files and return all of the records in the output data buffer parameter.
- Sort up to 32 input files, and return the output a set at a time through the QLGSRTIO API.
- Sort a set of records in the input data buffer parameter and produce 1 to 32 output files, each containing the full list of sorted records.
- Sort a set of records in the input data buffer parameter and return all of the records in the output data buffer parameter.
- Sort a set of records in the input data buffer parameter and return the output a set at a time through the QLGSRTIO API.
- Initialize a sort where the records are provided a set at a time through the QLGSRTIO API and produce 1 to 32 output files, each containing the full list of sorted records.
- Initialize a sort where the records are provided a set at a time through the QLGSRTIO API and are also returned a set at a time through that API.

The sort allows the application to indicate whether character data should be sorted as a binary (hexadecimal) sort or sorted using a national language sort sequence for obtaining sort results consistent with a specific language. The caller

can indicate if character data being sorted might contain double-byte (DBCS) characters. This type of field is considered a DBCS-open field, where DBCS data is surrounded by control characters. If a field is defined as such, the API will apply a national language sort sequence only to the single-byte data in the field. Performance may be slower when this type of character field is defined because of additional checking that occurs for DBCS data. If a field is defined as a single-byte field but DBCS data exists in the field, the national language sort sequence is applied to the DBCS data as if it were single-byte data, which may result in an unexpected sequence.

The sort also allows a character field to be defined so that the national language sort sequence will never be used. This definition is helpful if DBCS-graphic data is being sorted when there are no control characters surrounding the DBCS data. It can also be used to define character fields that contain hexadecimal control data.

The QLGSORT API requires that no earlier sort be active for this job at the time the sort is called. This is especially important when the QLGSORT API is called to initialize a sort and the QLGSRTIO API is repeatedly called to add data to the sort or return data from the sort. When retrieving data from the sort using the QLGSRTIO API, the normal operation is to continue calling QLGSRTIO until all of the data is returned. Whenever a get request is made and there is no more data to be returned, the sort is ended. If a get request is made and data still remains to be returned, the sort is still active until another get request is made and all of the data is returned. Another way to end the sort is to make a cancel request on the QLGSRTIO API call. The cancel request causes the sort to end immediately, regardless of whether there is still data to be returned. Whenever a sort is active and QLGSORT is called again by the same job, an error is returned.

When using the QLGSRTIO API to put data to the sort and the output is to be put in output files, the end put operation on the call to the QLGSRTIO API causes the data to be sorted and the output to be generated. The sort is then ended without the need to call QLGSRTIO with a cancel request.

When QLGSORT is called once to perform sorting of data in an input data buffer or in files, with the output going to an output data buffer or files, the sort is ended within that one call. All internal spaces have been deleted. Subsequent calls to the QLGSORT API are therefore valid.

If input files are specified and an error occurs with any of them, an error message is returned and no sorting occurs. It is up to the caller to determine if the QLGSORT API must be called again to perform the sort operation. There is no capability to continue with the previous sort. If output files are specified and an error occurs with any of them, the sort continues to put data to the files that are not in error. If all of the output files are in error, a message is returned indicating that the data was sorted but there were no valid output files in which to put the data.

Authorities and Locks

Input File Authority	*USE
Input File Library Authority	*USE
Output File Authority	*UPDATE
Output File Library Authority	*USE
Sort Sequence Table Authority	*USE
Sort Sequence Table Library Authority	*USE
Input File Lock	*SHRNUP
Output File Lock	*EXCL
Sort Sequence Table Lock	*SHRNUP

Required Parameter Group

Request control block
INPUT; CHAR(*)
Information defining the sort, such as whether a list of files or an input data buffer will be sorted. It also defines the keys to be used for the sort. Refer to "Format of Request Control Block" on page 41-7 for details.

Input data buffer
INPUT; CHAR(*)
The input data to be sorted. The calling program is responsible for adding all of the data to be sorted to this parameter before calling the QLGSORT API. The input data buffer parameter must contain the data to be sorted when the type of request field is 4, 5, or 6.

This parameter is ignored when the type of request field is 1, 2, 3, 7, or 8 because either the file data will be sorted or the data will be provided through calls to the QLGSRTIO API.

Output data buffer
OUTPUT; CHAR(*)
The sorted output data to be returned to the calling program. The output data buffer parameter will contain the sorted output data when the type of request field in the request control block is 2 or 5.

This parameter is ignored when the type of request field is 1, 3, 4, 6, 7, or 8 because data is either provided in output files or through calls to the QLGSRTIO API.

The QLGSORT API returns only the data that the buffer can hold, up to the length of data that is available to be returned. The size of the output data buffer must be greater than or equal to the record length field of the request control block. This is to ensure that at least one record can be provided as output. The output data buffer parameter can be the same as the input data buffer parameter.

Length of output data buffer
INPUT; BINARY(4)
The maximum length of the output data to be returned to the calling program in the output data buffer parameter. If this length is larger than the actual size of the output data buffer parameter, the results may not be predictable. The length of output data buffer must be at least

as large as the record length value specified in the request control block when the type of request field in the request control block is 2 or 5. If it is not, an error is returned.

The length of output data must be set to 0 when the type of request field is 1, 3, 4, 6, 7, or 8 because the data will be returned either as output files or through calls to the QLGSRTIO API.

Length of returned data
OUTPUT; BINARY(4)
The actual length of the output data added to the output data buffer parameter. When no output data is returned, this parameter is set to 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.

Format of Request Control Block

For a description of the fields in this format, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Length of request control block
4	4	BINARY(4)	Type of request
8	8	BINARY(4)	Reserved
12	C	BINARY(4)	Options
16	10	BINARY(4)	Record length
20	14	BINARY(4)	Record count
24	18	BINARY(4)	Offset to key list
28	1C	BINARY(4)	Number of keys
32	20	BINARY(4)	Offset to national language sort information
36	24	BINARY(4)	Offset to input file list
40	28	BINARY(4)	Number of input files
44	2C	BINARY(4)	Offset to output file list
48	30	BINARY(4)	Number of output files
52	34	BINARY(4)	Reserved
Note: Format of entries in the key list. The following fields are repeated for each key entry. The decimal and hexadecimal offsets depend on the number of key entries. The first key entry is found by using the offset to key list field. At least one key must be specified. Sorting will be done on the keys in the order they are specified.			
*	*	BINARY(4)	Key starting position
*	*	BINARY(4)	Key string size
*	*	BINARY(4)	Key data type
*	*	BINARY(4)	Sort order
Note: Format of national language sort information. The following fields are not repeated. The decimal and hexadecimal offsets depend on the number of key entries. The first field is found by using the offset to national language sort information field. If you want a hexadecimal sort instead of a national language sort, set the offset to national language sort information field to 0. In this case, you do not need to specify the sort sequence fields.			
*	*	CHAR(20)	Qualified sort table name
*	*	BINARY(4)	Sort sequence CCSID
*	*	CHAR(10)	Sort sequence language ID
*	*	CHAR(256)	Sort sequence table
Note: Format of entries in the input file list. The following fields are repeated for each input file entry. The decimal and hexadecimal offsets depend on the number of input file entries and the number of key entries. The first input file entry is found by using the offset to input file list field. If the input data buffer parameter contains data to be sorted, or data will be provided through the QGLSRTIO API, the offset to input file list field must be set to 0.			
*	*	CHAR(20)	Qualified input file name
*	*	CHAR(10)	Input member name
Note: Format of entries in the output file list. The following fields are repeated for each output file entry. The decimal and hexadecimal offsets depend on the number of input file list entries and key list entries. The first output file entry is found by using the offset to output file list field. If the sorted records are to be returned in the output data buffer, or data will be returned through calls to the QLGSRTIO API, the offset to output file list field must be set to 0.			
*	*	CHAR(20)	Qualified output file name
*	*	CHAR(10)	Output member name

Field Descriptions

Input member name. The name of the member in the input file. A value of *FIRST indicates that the first member of the file will be used. A value of *LAST indicates that the last member of the file will be used. If the specified member does not exist, an error is returned and no sorting is done.

Key data type. The following are the values that may be specified:

0 Signed binary.
1 Signed binary floating point.
2 Signed zoned decimal.
3 Signed packed decimal.
4 Character with national language sort sequence applied, if specified. DBCS data will be treated as single-byte data.
5 Mixed character with national language sort sequence applied, if specified, only to single-byte data.

| *6* Character with no national language sort sequence applied, if specified.
| *7* Unsigned packed decimal. All numbers will have the sign forced positive ('F'X).
| *8* Unsigned zoned decimal. All numbers will have the sign forced positive ('F'X).
| *9* Unsigned binary.
| *10* Zoned decimal with a sign over the leading digit. The sign occupies the left-most 4-bits of the value.
| *11* Zoned decimal with a separate trailing sign. Valid values are +, -, and blank.
| *12* Zoned decimal with a separate leading sign. Valid values are +, -, and blank.
| *13* Date in the form of MM/DD/YY, where:
| *YY* Year
| *MM* Month
| *DD* Day
| *14* Date in the form of DD/MM/YY
| *15* Date in the form of DD.MM.YYYY
| *16* Date in the form of DD/MM/YYYY
| *17* All other date/time formats
| *18* Time in the form of HH:MM xM, where:
| *HH* Hour
| *MM* Minute
| *x* A or P

| The use of any other value will cause an error to be | returned.

| **Key starting position**. The starting position of the key field | in the record. The starting position must be greater than 0 | and cannot exceed the record length specified in the record | length field, or an error is returned.

| The same key starting position is used for all records to be | sorted. Padding of blanks occurs on records that are shorter | than the record length field. If a key starting position is | within the area that is padded, these records are normally | sorted into the first positions.

| **Key string size**. The number of bytes to be used for sorting | this field. The key string size must be greater than 0, and | the string size plus the key starting position cannot exceed | the record length field; otherwise, an error is returned. Also, | the sum of the size of all of the keys cannot exceed 2000 | bytes. If the sum exceeds this maximum, an error is | returned.

| **Length of request control block**. The total length of the | request control block. The minimum size is 72 bytes, which | allows for one key in the key list, no input or output files, and | no national language sort sequence information. An error is | returned if the length specified is less than the minimum.

| **Number of input files**. The number of input files specified | in the input file list. If the number of input files is 0 and the | request type field is 1,2, or 3 (file sort), an error is returned. | Up to 32 input files can be specified.

| **Number of keys**. The number of keys specified in the | request control block. The sum of the lengths of all of the | key fields cannot exceed 2000 bytes. Therefore the number | of keys is limited to a maximum of 2000, which allows each | key to be 1 byte long. At least one key field must be | defined.

| **Number of output files**. The number of output files speci- | fied in the request control block. If the number of output files | is 0 and the request type field is 1, 4, or 7 (file output), an | error is returned. Up to 32 output files can be specified. | Each output file will contain exactly the same set of records.

| **Offset to input file list**. Offset to the start of the input file | list structure. If data in the input data buffer parameter is | being sorted, or input data will be provided by the | QLGSRTIO API, the input file list structure is not required | and this offset must be set to 0.

| **Offset to key list**. Offset to the start of the key list struc- | ture. At least one key must be defined in the key list struc- | ture, or an error is returned.

| **Offset to national language sort information**. Offset to | the start of the national language sort information structure. | If you want a hexadecimal sort instead of a national lan- | guage sort, set this offset to 0. In this case, you do not need | to specify the sort sequence fields.

| **Offset to output file list**. Offset to the start of the output | file list structure. If data is being returned in the output data | buffer parameter or if the QLGSRTIO API will be used to | return the sorted data, the output file list structure is not | required and this offset must be set to 0.

| **Options**. Additional options to be handled by the sort. The | following values are defined:

| *0* No options used.
| *1* Log record count messages. One message is returned in the job log for each output file indicating the number of records written to the output file. When output files are not used, one message is returned indicating the total number of records sorted.
| *2* Throw away duplicate keyed records. The normal processing is to maintain duplicates in the order in which they are received. When putting output data to a database file in which the file has been defined to have unique keys, a message is returned when a duplicate key is encountered and no further data is entered into the file or any other file. This option allows for succeeding data to be written to the file by bypassing records with duplicate keys.
| *3* Log record count messages and throw away duplicate keyed records. This option is a combination of options 1 and 2.

| **Output member name**. The name of the member in the | output file. A value of *FIRST indicates that the first member | of the file should be used. A value of *LAST indicates that | the last member of the file should be used.

| **Qualified input file name**. The name of a file whose data | is to be sorted, 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.

The library name can have the following special values:

*LIBL The current library list.
*CURLIB The job's current library.

All values must be padded with blanks to the right of the value.

If a file is specified that does not exist or is unavailable (for example, locked), an error is returned for that file and no further files are read. No sorted data is returned for any of the files already read without errors.

The QLGSORT API supports database (both logical and physical), diskette, and tape files. Any other file type is not recognized and an error is returned. Variable length records and variable length fields are not supported.

The data in each of the files is sorted as the common record length specified in the record length field. The data is truncated or padded with blanks as necessary when the record length of the input file is different than the record length value.

The total length of all the records in all the input files cannot exceed 16MB for the data to be returned in the output data buffer parameter. If your files have more data than this, you should consider having the output data sent to output files, or use the QLGSRTIO API to repeatedly retrieve sets of sorted data.

Logical files can be used as input files. Performance may be improved when using logical files because the index of the logical file is used.

Qualified output file name. The name of a file that is to receive sorted data, 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.

The library name can have the following special values:

*LIBL The current library list.
*CURLIB The job's current library.

All values must be padded with blanks to the right of the value.

If a file is specified that does not exist or is unavailable (for example, locked), a diagnostic error is returned indicating the error and the file in error. The QLGSORT API will continue to put data to the other files that do not have errors. If none of the output files could be opened, an error message is returned indicating that the sort was successful but there were no files in which the sorted data could be returned.

The QLGSORT API supports database (both logical and physical), diskette, and tape files. Any other file type is not recognized and an error is returned, but the sort will continue to provide output to any other output files that are supported. Variable length records and variable length fields are not supported.

If the sorted output is returned in the output data buffer parameter, or returned using the QLGSRTIO API, the output file list is not used. If the number of output files field is greater than 0, there must be exactly that number of output files specified, or an error is returned.

Each output file contains the full sorted set of records. Each sorted record has a common record length as defined in the record length field. Record truncation or padding of blanks occurs when writing to the output file and when the file record length is different than the sorted record length. One reason to have more than one output file is to provide a backup or shadow of your output data as it is being sorted.

Qualified sort table name. The name of a sort table to be used for sorting the character data, and the library in which it is located. The first 10 characters contain the sort table name, and the second 10 characters contain the library name.

When a qualified sort table name is specified, the sort sequence language ID and sort sequence table fields are ignored.

The following special values are supported for the table name. These values must be in uppercase, left-justified, and padded with blanks to the right of the value.

*JOB The sort sequence of the job.
*JOBRUN The API treats this value the same as *JOB.
*LANGIDUNQ
The unique-weighted sort sequence table that is associated with the sort sequence language ID field.
*LANGIDSHR
The shared-weighted sort sequence table that is associated with the sort sequence language ID field.
*HEX The hexadecimal value of the characters.
*TABLE The sort sequence table provided in the sort sequence table field.

Note: Both *JOB and *JOBRUN are accepted to accommodate calls from other programs that accept both values and for which the two have different semantics.

The following special values are supported for the library name. These values must be in uppercase, left-justified, and padded with blanks to the right of the value.

*LIBL The user's library list.
*CURLIB The current library.

If a special value is used for the table name, the second 10 characters used for the library name must be blank.

Record count. The number of records that exist in the input data buffer parameter when input data is to be sorted. Only the number of records specified in the record count parameter will be sorted, even if more than this number exist in the input data buffer parameter.

The record count field must be greater than 0 when the type of request field is 4, 5, or 6 (input data to be sorted). If it is not, an error is returned.

The record count field must be set to 0 when the type of request field is 1, 2, 3, 7, or 8 because input files are being sorted or the data will be provided by the QLGSRTIO API.

Record length. The record length that the sort will use for processing. When an input data buffer is being sorted, each record in the buffer is defined to be this length. When a list of files is being sorted, this record length defines the length used for sorting all the files.

If an input file has a record length that is longer than the specified record length, it is truncated. If an input file has a record length that is shorter, it is padded with blanks.

When data is being provided through the QLGSRTIO API, this record length defines the length used for sorting all the records. Each set of records added to the sort through the QLGSRTIO API defines a record length for that set. Records that are longer than the record length defined here are truncated. Records that are shorter are padded with blanks.

The following special value is defined:

0 QLGSORT assumes the maximum record length to be that of the first input file processed. If 0 is specified when an input data buffer is being sorted, or if data is being provided through calls to the QLGSRTIO API, an error occurs.

Reserved. A reserved field. This field must be set to 0.

Sort order. The order to be used for sorting.

The following special values are defined:

1 Sort in an ascending sequence.
2 Sort in a descending sequence.

Sort sequence CCSID. The CCSID to be used, along with the sort sequence language ID, for retrieving the national language sort sequence table for sorting the character data.

The valid values for this parameter are:

0 The CCSID of the job is the CCSID of the sort sequence table to be used.
1-65533 A valid CCSID in this range must be specified or an error is returned.
65535 The CCSID of the sort sequence table that is found should be used. No CCSID conversion should be done on the found table.

The following table summarizes when the CCSID value is required and when it will be ignored based on the value in the qualified sort table name field in the request control block:

Qualified Sort Table Name	CCSID
*JOB	Required
*JOBRUN	Required
*LANGIDUNQ	Required
*LANGIDSHR	Required
*HEX	Ignored
*TABLE	Ignored
Table name and library	Required

Sort sequence language ID. The language ID to be used to obtain a national language sort sequence table for sorting the character data. All values must be padded with blanks to the right of the value and must be in uppercase.

Possible values for this parameter are:

**JOB* The language identifier of the job.
**JOBRUN* The API treats this value the same as *JOB.
Specific language identifier
A valid 3-character language identifier. For example, Danish would be "DAN." See the *National Language Support Planning Guide* for a complete list of valid language identifiers.

Note: Both *JOB and *JOBRUN are accepted to accommodate calls from other programs that accept both values and for which the two have different semantics.

The following table summarizes when the sort sequence language ID is required and when it is ignored based on the value in the qualified sort table name field in the request control block:

Qualified Sort Table Name	Language ID
*JOB	Required
*JOBRUN	Required
*LANGIDUNQ	Required
*LANGIDSHR	Required
*HEX	Ignored
*TABLE	Ignored
Table name and library	Ignored

Sort sequence table. The sort table to be used for sorting the character data. This field is only used when the qualified sort table name field is *TABLE. Otherwise, it is ignored.

Type of request. The type of sort operation being requested. The following values are defined:

1 Input files are sorted, and the output goes to an output file.
2 Input files are sorted, and the output goes to the output data buffer parameter.
3 Input files are sorted, and the output is held for the QLGSRTIO API to retrieve one set at a time.
4 Data in an input data buffer is sorted, and the output goes to an output file.
5 Data in an input data buffer is sorted, and the output goes to the output data buffer parameter.
6 Data in an input data buffer is sorted, and the output is held for the QLGSRTIO API to retrieve one set at a time.
7 Input data is provided by the QLGSRTIO API, and the output goes to an output file.

8 A sort is initialized so that data can be provided through the QLGSRTIO API one set at a time and retrieved one set at a time through the QLGSRTIO API.

The following table summarizes the input source and output target for each type of request:

Type of Request	Source	Target
1	Input files	Output files
2	Input files	Output data buffer
3	Input files	QLGSRTIO API calls
4	Input data buffer	Output files
5	Input data buffer	Output data buffer
6	Input data buffer	QLGSRTIO API calls
7	QLGSRTIO API calls	Output files
8	QLGSRTIO API calls	QLGSRTIO API calls

Error Messages

CPF2115 E Object &1 &2 type *&3 damaged.
CPF24B4 E Severe error occurred while addressing parameter list.
CPF3BC6 E Sort sequence &1 not valid.
CPF3BC7 E CCSID &1 outside of valid range.
CPF3BC8 E Conversion from CCSID &1 to CCSID &2 is not supported.
CPF3BC9 E Conversion from CCSID &1 to CCSID &2 is not defined.
CPF3BCC E Language identifier &1 not valid.
CPF3BD5 E Sort request control block field &1 is not valid.
CPF3BD6 E Key field &1 is not valid for key number &2.
CPF3BD7 E Key size exceeds maximum.
CPF3BD8 E Qualified input file name or member is not valid for input file &3.
CPF3BD9 E Qualified output file name or member is not valid for output file &3.
CPF3BDA E No output files could be opened.
CPF3BDB E Not enough space to sort all records.
CPF3BDC E Duplicate keys encountered for file &1 in library &2.
CPF3CF1 E Error code parameter not valid.
CPF3BD3 E Sort already active.
CPF9504 E Object &1 in library &2 damaged.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Not able to allocate &1 in &2.
CPF9810 E Library &1 not found.
CPF9830 E Library in use.
CPF9845 E Error occurred while opening file &1.
CPF9820 E Not authorized to use library &1.
CPF9846 E Error while processing file &1 in library &2.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Sort Input/Output (QLGSRTIO) API

Parameters

Required Parameter Group:

1	Request control block	Input	Char(16)
2	Input data buffer	Input	Char(*)
3	Output data buffer	Output	Char(*)
4	Length of output data buffer	Input	Binary(4)
5	Output data information	Output	Char(16)
6	Error code	I/O	Char(*)

The Sort Input/Output (QLGSRTIO) API is used to provide a set of records at a time to be sorted and to return a set of records at a time that have already been sorted. This API can only be used after the QLGSORT API has been called to initialize the sort function. This initialization defines such things as the length of data to be sorted and the fields to be sorted. An error is returned if QLGSRTIO is called without QLGSORT having been called for initializing.

After the initialization is complete, the QLGSRTIO API can be called repeatedly to add a set of records to be sorted. When all of the records have been provided, the QLGSRTIO API must be called with an end put request, which causes the added records to be sorted. Once the records are sorted, the QLGSRTIO API can be called repeatedly to return sets of sorted records until no more records can be returned or the application has determined that it no longer wants sorted records.

The QLGSRTIO API can also be used to provide sets of records at a time to the sort and have the output from the sort go to output files. The type of request in the request control block on the QLGSORT API can be set to specify that the output from the sort is output files.

The QLGSRTIO API provides a function to cancel the sort at any time. When the sort is canceled, all work areas initialized for the sort are deactivated, so the QLGSORT API can be called again to perform another sort. If the QLGSRTIO API is being used to return records from a sort a set at a time, the QLGSRTIO API automatically ends the sort when a get request is made and all of the sorted records have been returned. If all of the records are not retrieved, a cancel is required to complete this sort if another call to the QLGSORT API will be made within this job. If a cancelation is not requested, the sort is still considered active and an error is returned on any future calls to the QLGSORT API within this job. When the job ends, the sort automatically ends. If the QLGSRTIO API is only being used to add records to the sort a set at a time, with the output going to output files, the end put request causes the data to be sorted and put to the output files, and the sort ends. No additional cancel request is needed.

Authorities and Locks

The locks needed for this API are set in the QLGSORT API and remain in effect until a cancel request is made by the QLGSRTIO API or until an end put request is made to return the sorted data to output files.

Required Parameter Group

Request control block
INPUT; CHAR(16)
Whether data is being put to the sort or retrieved from the sort. This parameter also defines information about the input and output data. See "Format of Request Control Block" for details.

Input data buffer
INPUT; CHAR(*)
The data to be added to the sort. This parameter is only used on a put request. It is ignored for end put, get, and cancel requests. If the call to QLGSORT specified that input files or an input data buffer were to be sorted and if QLGSRTIO is then called specifying a put request and input data, an error is returned.

Output data buffer
OUTPUT; CHAR(*)
The sorted data being returned to the application. This parameter is only used on a get request. It is ignored for put, end put, and cancel requests. If the call to the QLGSORT API specified that files were to be returned and the QLGSRTIO API is then called specifying a get request, an error is returned. This parameter can be the same as the input data buffer parameter.

Length of output data buffer
INPUT; BINARY(4)
The maximum amount of data that can be returned from the sort on a get request. This value must be greater than 0 or an error is returned. QLGSRTIO returns only the number of records specified in the request control block parameter, but only up to the size of the output data buffer. The length of the output data buffer must be at least as large as the record length value specified in the request control block on the QLGSORT API call. That length specified the record length to be used by the sort. If this output data length is not large enough to hold at least one record, an error is returned.

Output data information
OUTPUT; CHAR(16)
Information to be returned to the calling program as a result of a put or get request. See "Format of Output Information" for details on the 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.

Format of Request Control Block

For a description of the fields in this format, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Type of request
4	4	BINARY(4)	Reserved
8	8	BINARY(4)	Record length
12	C	BINARY(4)	Record count

Format of Output Information

For a description of the fields in this format, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Number of records processed
4	4	BINARY(4)	Number of records available
8	8	CHAR(8)	Reserved

Field Descriptions

Number of records available. For a put request, the total number of records that have been put (are available) for sorting. For the first put request, this will equal the number of records processed. After the second put request, it will contain the total of the first two put requests. An application might find this helpful if it only wants to sort a maximum number of records without counting the number from each put.

For a get request, this field defines the total number of records that have not been returned to the application. An application can use this information to determine an appropriate time to end if it does not intend to process all of the sorted records.

Number of records processed. For a put request, the number of records actually put to the sort function from the input data buffer parameter. This value is always the same as the record count specified in the request control block.

For a get request, this field defines the number of records actually returned in the output data buffer parameter. This value never exceeds the record count specified in the request control block parameter. If more records are requested than are available, the records processed count will be set to the actual number returned.

Record count. For a put request, the number of records that are being provided to the sort. The record count must be greater than 0 or an error is returned. Only the number of

records specified by this field are processed even if more records are provided in the input data buffer parameter.

For a get request, this field defines the maximum number of records that can be returned to the application. The number of records processed field in the output data information parameter indicates the actual number of records returned.

Using record counts greater than 1 can improve performance.

Record length. The size of each record given to the sort or the size of each record returned from the sort. For a put request, each record retrieved from the input data buffer must have the length specified by the record length field. The initialization done when calling the QLGSORT API indicated the maximum record length to be used by the sort program. If the input data record length is shorter than the sort record length, each of the records is padded with blanks. If the input data record length is longer than the sort record length, each input record is truncated.

On a get request, each record is put to the output data buffer parameter with the specified record length. If the sort record length is longer, each record is truncated. If the sort record length is shorter, each record is padded with blanks.

Reserved. A reserved field. This field must be set to 0 in the request control block. In the output information, the reserved character field is set to blanks.

Type of request. The type of operation being requested. The following values are defined:

1 (Put)
Input data is provided to the sort function where it is stored by the sort program in internal buffers until an end put request is made and the data is sorted. One or more calls to the API with the put request can be done until all of the data to be sorted has been provided to the sort program.

A put request is only valid after a call to the QLGSORT API is done to initialize a sort for input from QLGSRTIO and before an end put request is made. If a put request is made at any other time, an error is returned.

2 (End put)
All of the input data has been added to the sort, and the data should now be sorted. This operation must be requested before a get request is made, or an error is returned.

An end put request causes the input to the sort to be complete. No additional put requests can be made to the QLGSRTIO API without first doing an initialization through the QLGSORT API; otherwise, an error is returned.

3 (Get)
Sorted data is returned to the caller. One or more calls to the API with the get request can be done until all of the data has been retrieved or no further data is needed. If no further get requests are made but data is still available to be retrieved, a cancel request must be made to end the sort.

A get request is valid for each of the following conditions:

- After an end put request was done (to cause the data to be sorted)
- After a previous get request
- If the sort was done on input files or on an input data buffer using sort request type 3 or 6 of the QLGSORT API

If a get request is made at any other time, an error is returned.

4 (Cancel)
The sort should be canceled immediately. All internal buffers are cleared and deleted. This operation can be requested at any time. If requested after a put request, no sorting is performed. If requested after a get request, no further sort records are returned. The sort is automatically ended when a get request is made and no data is available to be returned. If all of the data was not retrieved, a cancel request should always be made. If this is not done and another sort request is made using the QLGSORT API, an error occurs because a sort is already active.

Error Messages

CPF24B4 E Severe error occurred while addressing parameter list.
CPF3BD0 E Request control block field &1 is not valid.
CPF3BD1 E Output data length parameter is not valid.
CPF3BD2 E Type of request &1 is not valid at this time.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Truncate Character Data (QLGTRDTA) API

Parameters

Required Parameter Group:

1	Output data buffer	Output	Char(*)
2	Length of output data returned	Output	Binary(4)
3	Remaining data buffer	Output	Char(*)
4	Remaining data length	Output	Binary(4)
5	Input data buffer	Input	Char(*)
6	Length of buffers	Input	Binary(4)
7	Truncate length	Input	Binary(4)
8	CCSID of input data	Input	Binary(4)
9	Error code	I/O	Char(*)

The Truncate Character Data (QLGTRDTA) API truncates a CCSID-tagged string of character data to a requested length. This API is used to truncate a string of data properly. The string is truncated with respect to the encoding of the data using the encoding scheme of the CCSID.

A typical use of this API would be to properly truncate mixed-byte strings for use in a display screen.

This API works in the following manner:

1. Given the truncate length requested for the result, truncate the input data in a proper manner[1] to no longer than this value. If the length of the input data buffer is less than the requested truncate length, the returned length is set to the input length and the data in the input data buffer is placed in the resultant output data buffer.
2. Place this result into the requested output data buffer and set its length in the length of output data returned parameter. Pad the rest of this parameter with the appropriate blank characters. When the length of input is less than the truncate length, all data is placed in the output data buffer because the condition would call for no truncation.
3. Place the remaining input data (properly formed) into the remaining data buffer and set its length in the remaining data length parameter. Properly formed data refers to providing a valid string for the type of data being used. For example, mixed EBCDIC data must have matched control characters (X'0E' and X'0F') around any double-byte character data.

Required Parameter Group

Output data buffer
OUTPUT; CHAR(*)
The buffer that receives the properly formed truncated string. This value must be the same size as the value specified for the input data buffer parameter.

Length of output data returned
OUTPUT; BINARY(4)
The number of bytes of data actually returned in the resultant output buffer.

Remaining data buffer
OUTPUT; CHAR(*)
The buffer that receives the data remaining after the truncation has been done. This buffer must be the same size as the input data buffer.

Remaining data length
OUTPUT; BINARY(4)
The length of the data remaining after the truncation has been done. This data is the length of the remaining data buffer.

Input data buffer
INPUT; CHAR(*)
The buffer that holds the input data.

Length of buffers
INPUT; BINARY(4)
The length of each of the following data buffers:

- Input data buffer
- Output data buffer
- Remaining data buffer

The maximum size of each data buffer is 32767 bytes.

Truncate length
INPUT; BINARY(4)
The maximum length of data to be returned in the output data buffer. Valid values are 1 through 32767.

CCSID of input data
INPUT; BINARY(4)
The CCSID of the data to be truncated. Valid values are 1 through 65533 and 65535.

If the CCSID tag of the data is not known, a value of 65535 should be used. If the CCSID value is 65535, the data is assumed to be a mixed-byte EBCDIC string that is properly formed.

The only supported encoding schemes are single-byte data, mixed-byte EBCDIC, and double-byte data. See the *National Language Support Planning Guide* for more information on CCSID values and encoding schemes.

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 occurred while addressing parameter list.
CPF2647 E Buffer length not valid.
CPF3BC7 E CCSID &1 outside of valid range.
CPF3BCA E CCSID &1 not supported.
CPF3BCB E Encoding scheme &1 of CCSID &2 not supported.
CPF3BCF E Truncate length not valid.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Validate Language ID (QLGVLID) API

[1] **Proper manner** means that no characters are split and no control information is lost. In the situation where characters are split, the truncation point is readjusted to an acceptable position to allow truncation. If control characters are required, a shift-in character is added to the end of the result string and a shift-out character is added to the beginning of the remaining data.

Parameters

Required Parameter Group:

1	Language ID to validate	Input	Char(3)
2	Error code	I/O	Char(*)

The Validate Language ID (QLGVLID) API validates a language identifier to ensure it is supported. If the language Identifier is valid, no error is returned.

Required Parameter Group

Language ID to validate
INPUT; CHAR(3)
The language identifier to validate. The identifier must be in uppercase.

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

CPF1F41 E Severe error occurred while addressing parameter list.
CPF3BCC E Language identifier &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 13. Network Management APIs

Chapter 42. 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:

- APPN* topology APIs

  Deregister APPN Topology Information (QNMDRGTI)

  Register APPN Topology Information (QNMRGTI)

- SNA management services transport APIs

  Change Mode Name (QNMCHGMN)
  Deregister Application (QNMDRGAP)
  End Application (QNMENDAP)
  Receive Data (QNMRCVDT)
  Receive Operation Completion (QNMRCVOC)
  Register Application (QNMREGAP)
  Retrieve Mode Name (QNMRTVMN)
  Send Error (QNMSNDER)
  Send Reply (QNMSNDRP)
  Send Request (QNMSNDRQ)
  Start Application (QNMSTRAP)

- Alert APIs

  Generate Alert (QALGENA)
  Retrieve Alert (QALRTVA)
  Send Alert (QALSNDA)

- Node list API

  List Node List Entries (QFVLSTNL)

- Problem log APIs

  Filter Problem (QSXFTRPB)
  Work with Problem (QPDWRKPB)

- Registered filter APIs

  Deregister Filter Notifications (QNMDRGFN)
  Register Filter Notifications (QNMRGFN)
  Retrieve Registered Filters (QNMRRGF)

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.

APPN Topology Information APIs

APPN topology information APIs allow an application to obtain information about the current APPN topology, and to register and deregister for information about ongoing updates to the topology. Current topology information is an option provided by the Register for APPN Topology Information (QNMRGTI) API. When this option is requested, the current APPN topology is reported to a user space specified by the application running the API. This API also provides topology update options that allow the application to register a queue to receive information about specific types of APPN topology updates. Topology updates are reported asynchronously to the specified queue as they occur in the network.

A queue remains registered for topology updates until one of the following occurs:

- The queue is deregistered by the application using the Deregister APPN Topology Information (QNMDRGTI) API.
- An error is encountered enqueueing topology updates, forcing automatic deregistration of the queue by the system.
- The application's job is ended causing registration to be cleaned up.
- An IPL is performed causing registration to be cleaned up.

One application in each job on the system may register one queue for topology updates. Multiple queues may not be registered for topology updates within the same job.

The specific types of topology updates that an application may register to receive are:

- Local end node (*EN) updates
- Local virtual node (*VN) updates
- Local network node (*NN) updates
- Network network node (*NN) updates
- Network virtual node (*VN) updates

The queue and user space objects specified on the APPN topology information APIs must be managed entirely by the application; for example, the application must create, delete and maintain the objects itself, using the APIs for those objects. The application is responsible for any error handling should these objects become damaged or deleted.

If an error occurs while reporting the current topology to the specified user space, an error is returned through the API. If an error occurs while enqueueing ongoing topology updates to a registered queue, the resulting error messages are sent to the job log, followed by diagnostic message CPD91C9, and the queue is automatically deregistered by the system. If automatic handling of this diagnostic error message is necessary, an application could periodically scan the job log for this message using the List Job Log (QMHLJOBL) API and take appropriate action.

After the current topology has been requested using the QNMRGTI API, the data returned in the user space may be retrieved using the Retrieve User Space (QUSRTVUS) API. If a data queue is registered for topology updates using the QNMRGTI API, topology update records may be retrieved out of a data queue using the Receive Data Queue (QRCVDTAQ) API. If a user queue is registered (rather than a data queue), the application must use the dequeue (DEQ) MI instruction to retrieve queue records. The first 10 characters of each queue entry contains the value *APPNTOP so that the application can distinguish these records from others on the queue. This allows a queue to be used for multiple purposes.

Local and Network Topology Updates: Topology updates can be separated into two classes:

- Network topology updates
- Local topology updates

Local topology updates can be reported about an end node or network node system, but network topology updates can be reported only about a network node system.

APPN Network Topology Updates: APPN network topology identifies the following in an APPN subnetwork:[1]

- All network nodes and virtual nodes in the subnetwork
- Transmission groups interconnecting network nodes and virtual nodes in the subnetwork
- Transmission groups from network nodes in the subnetwork to network nodes in adjacent subnetworks

APPN network nodes exchange network topology updates in a subnetwork through topology database updates (TDUs). Therefore, only network nodes can report network topology updates. See *System Network Architecture Formats* for details about TDUs.

APPN Local Topology Updates: The local topology for an APPN node consists of the following:

- The local node
- Adjacent nodes (network nodes, end nodes, or virtual nodes to which the local node has a direct connection)
- Transmission groups from the local node to adjacent nodes

Both end nodes and network nodes can report local topology updates.

Adjacent Subnetworks: Network nodes in disconnected subnetworks may be connected by an intersubnetwork transmission group.[2] In this case, the network node at each end point of the transmission group is present only in the partner network node's local topology, not in its network topology.

For example, consider two network nodes with different network IDs in disconnected subnetworks:

- A network node with the following control point name and network ID: CPNAME=NN1, NETWORK ID=A
- A network node with the following control point name and network ID: CPNAME=NN2, NETWORK ID=B

When NN1 and NN2 are connected by an intersubnetwork transmission group, NN2 is present only in the NN1 local topology; it is not present in the NN1 network topology or other nodes in network A. This is because TDUs for NN2 are not exchanged in network A.

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.

---

[1] An APPN subnetwork consists of nodes having a common network ID and the links connecting those nodes.

[2] A group of links between directly attached nodes of 2 or more subnetworks appearing as a single logical link for routing messages.

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(4)		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 42-1 on page 42-4 shows how these SNA management services transport APIs work together.

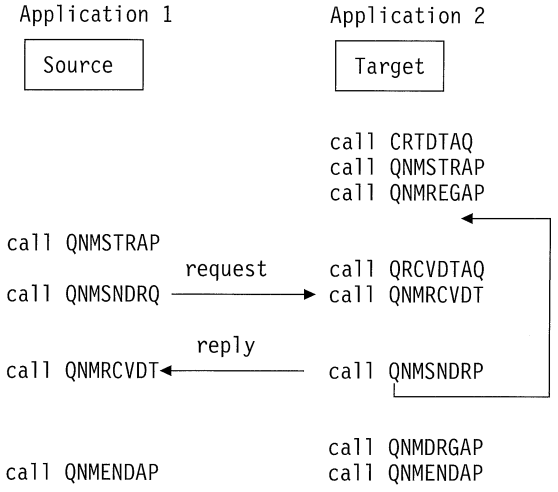


Figure 42-1. Applications Using SNA Management Services Transport APIs

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 42-2, which shows the flow of data requests and replies depending on the parameters specified.

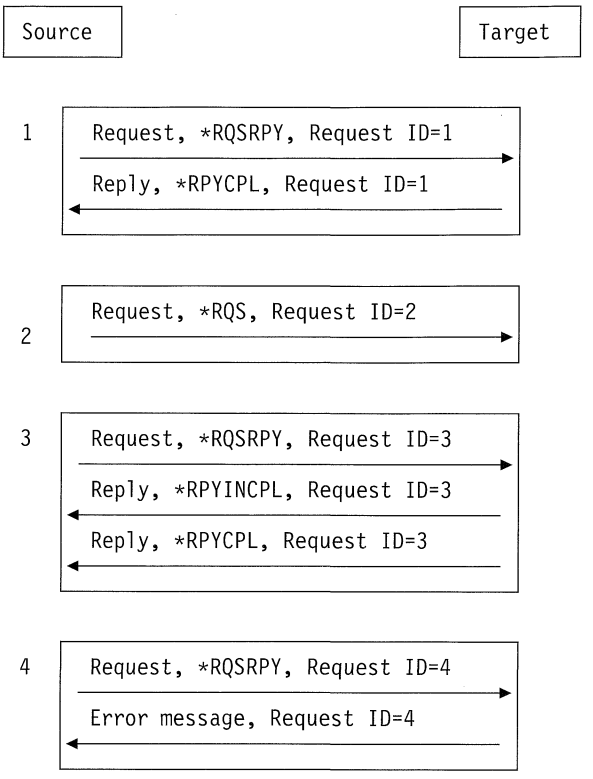


Figure 42-2. Data Types Handled by SNA Management Services Transport APIs

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 per-

forms 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. This is known as direct routing to 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.

Node List API

A node list contains a list of systems identified by a network ID and a control point name. The List Node List Entries (QFVLSTNL) API returns, in a user space, a list of the nodes contained in the specified node list object.

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 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.

Registered Filter APIs

A **filter** is a function you can use to assign events into groups and to specify actions to take for each group. The registered filter APIs allow a product to register a filter with the operating system. The product can receive notification of events recorded in a data queue by using the Send to Data Queue (SNDDTAQ) action of the Work with Filter Action Entry (WRKFTRACNE) command.

A product can use registered filter APIs for the following purposes:

- To register multiple filters at the same time for each event type (alert or problem log)
- To deregister a filter when notifications from that filter are no longer necessary
- To retrieve all the filters that are registered

The event notification record for a registered filter differs from notification records for other types of filters. The registered notification contains a common header for all events, as well as specific information based on the type of event. The common header includes the name of the notification, a function type, a format, the filter name and library, the group name, and a timestamp. The specific information for the problem log includes the problem ID, the last event logged, and the timestamp for the last event.

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)
The unique identifier for this application, which was returned by the Start Application (QNMSTRAP) API.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 42-7 for more information.

Required Parameter Group

Handle
INPUT; BINARY(4)
The unique identifier for this application, which was returned by the Start Application (QNMSTRAP) 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 parameter not valid.
CPF7ADC E Internal processing error.
CPF7AEE E Application &2 not registered.
CPF7AE2 E Handle &1 not found.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Deregister APPN Topology Information (QNMDRGTI) API

Parameters

Required Parameter Group:

1	Queue handle	Input	Binary(4)
2	Error code	I/O	Char(*)

The Deregister APPN Topology Information (QNMDRGTI) API causes the queue associated with the specified queue handle to be deregistered for APPN topology information. When a queue is deregistered, future topology updates are not reported to the queue.

The queue handle specified by this API must match the queue handle that was returned when the queue was registered with the Register APPN Topology Information (QNMRGTI) API.

Required Parameter Group

Queue handle
INPUT; BINARY(4)
A variable that represents a registered queue. This value was returned by the QNMRGTI API when the queue was 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.
CPF91C3 E Internal processing error.
CPF91C6 E Queue handle &1 not found.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Deregister Filter Notifications (QNMDRGFN) API

Parameters

Required Parameter Group:

1	Qualified filter name	Input	Char(20)
2	Filter type	Input	Char(10)
3	Error code	I/O	Char(*)

The Deregister Filter Notifications (QNMDRGFN) API deregisters a filter. If the identified filter is not currently registered, an error is returned.

Authorities and Locks

Registered Filter *USE
Registered Filter Library *USE

Required Parameter Group

Qualified filter name
INPUT; CHAR(20)
The qualified filter name that is being deregistered. The first 10 characters are the filter name, and the last 10 characters are the library name. Special values *LIBL and *CURLIB are not supported for the library name.

Filter type
INPUT; CHAR(10)
The filter type that is being deregistered. Special values supported are:

***ALR** Alert filter
***PRB** Problem log filter

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.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF91D1 E Filter &1/&2 of type &3 was not deregistered.
CPF91D2 E Filter type &3 not correct for this operation.
CPF91D4 E Filter &1/&2 of type &3 is not registered.
CPF9800 E All CPF98xx messages could be signaled. xx is from 01 to FF.

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)
The unique identifier for this application, which was returned by the Start Application (QNMSTRAP) 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 parameter not valid.
CPF7AE2 E Handle &1 not found.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.
CPF8160 E &8 damage on &4 type &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Generate Alert (QALGENA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Size of Alert	Output	Binary(4)
4	Qualified alert table (message file) name	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 *System Network Architecture Formats*. 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.

Size of Alert
OUTPUT; BINARY(4)
The size of the successfully generated alerts. If the size of the alert is smaller than the receiver variable length, the alert is returned in the receiver variable. If the size of the alert is greater than the receiver variable length, the alert is not returned.

Qualified alert table (message file) name
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 of the Display Message Description (DSPMSGD) command 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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Node List Entries (QFVLSTNL) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified node list name	Input	Char(20)
4	Error code	I/O	Char(*)

The List Node List Entries (QFVLSTNL) API returns, in a user space, a list of the nodes contained in the specified node list object.

Authorities and Locks

Node List Object Authority
*USE
Node List Object Lock
*SHRRD
User Space Authority
*CHANGE
User Space Lock *EXCLRD

Required Parameter Group

Qualified user space name
INPUT; CHAR (20)
The name of the user space that is to receive the generated list. The first 10 characters contain the user space name. The second 10 characters contain the name of the library where the user space is located. The following special values can be used for the library name:

***CURLIB**	The current library.
***LIBL**	The library list.

Format name
INPUT; CHAR(8)
The format of the data placed in the user space. The valid value is:

NODL0100	All node list entry information returned. This format is explained in "NODL0100 List Data Section."

Qualified node list name
INPUT; CHAR(20)
The name of the node list object from which the entries are to be retrieved. The first 10 characters contain the node list name. The second 10 characters contain the name of the library where the node list is located. Special values for the name of the node list library are:

***CURLIB**	The 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 Lists

The returned user space will contain:

- 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 a detailed description of each field in the information returned, see "Field Descriptions."

Input Parameter Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format name specified
28	1C	CHAR(10)	Node list object name specified
38	26	CHAR(10)	Node list object library name specified

Header Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library name used
20	14	CHAR(10)	Node list object name used
30	1E	CHAR(10)	Node list object library name used

NODL0100 List Data Section

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Node type
1	1	CHAR(20)	Node name
21	15	CHAR(50)	Text description
71	47	CHAR(1)	Reserved
72	48	BINARY(4)	Text description CCSID

Field Descriptions

Format name specified. The format name specified as input for the API.

Node list object library name specified. The name of the node list object library as specified on the call to the API.

Node list object library name used. The actual name of the node list object library used to report data.

Node list object name specified. The node list object name as specified on the call to the API. The node list object name contains a list of SNA nodes identified by an APPN network ID and control point name. The system-recognized identifier for the object type is *NODL.

Node list object name used. The actual node list object name used to report data. The node list object name contains a list of SNA nodes identified by an APPN network ID

| and control point name. The system-recognized identifier for
| the object type is *NODL.

| **Node name**. The name of a system in a network. If the
| node type equals 1, the node name consists of an
| 8-character network ID, an 8-character control point name,
| and 4 reserved characters.

| **Node type**. A value of 1 in this field identifies an SNA
| APPN node type.

| **Reserved**. An ignored field.

| **Text description**. The text description of the node list entry.

| **Text description CCSID**. The coded character set identifier
| (CCSID) used for the text description.

| **User space library name specified**. The user space
| library name as specified on the call to the API.

| **User space library name used**. The actual user space
| library name used to report data.

| **User space name specified**. The user space name as
| specified on the call to the API.

| **User space name used**. The actual user space name used
| to report data.

Error Messages

| CPF24B4 E Severe error while addressing parameter list.
| CPF3C21 E Format name &1 is not valid.
| CPF3CF1 E Error code parameter not valid.
| CPF813E E Node list &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.
| 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.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

Receive Data (QNMRCVDT) API

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)
| The unique identifier for this application, which was
| returned by the Start Application (QNMSTRAP) API.

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
| actually returned. The bytes available field specifies the
| total length of data available to be returned.

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.
| CPF7AEA E Application on control point &3.&4 failed.
| 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.
| CPF7AE3 E Management services transport operation not permitted.
| 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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

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)
The unique identifier for this application, which was returned by the Start Application (QNMSTRAP) API.

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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

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 42-27 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)
| The unique identifier for this application, which was
| returned by the Start Application (QNMSTRAP) API.

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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Register APPN Topology Information (QNMRGTI) API

Parameters

Required Parameter Group:

1	Queue handle	Output	Binary(4)
2	Options	Input	Array of Char(10)
3	Number of options	Input	Binary(4)
4	Format	Input	Char(8)
5	Qualified user space name	Input	Char(20)
6	Qualified queue name	Input	Char(20)
7	Queue type	Input	Char(10)
8	Replace registration	Input	Char(10)
9	Error code	I/O	Char(*)

The Register APPN Topology Information (QNMRGTI) API causes the requested APPN topology information to be reported. The application calling this API may request the current copy of the entire database (to be reported to the specified user space), or may register for information about particular types of ongoing updates to the topology (to be reported to the registered queue), or both.

A queue handle is returned by this program when a queue is successfully registered for topology updates. The queue handle identifies a registered queue, and must be used when the Deregister APPN Topology Information (QNMDRGTI) API is called. The queue handle is unique to a specific job.

When a queue is registered for ongoing topology updates, the specified types of updates for which the queue is registered will be asynchronously enqueued on an ongoing basis. If current topology is also requested on the application calling this API, the current topology is reported to the user space before topology updates are reported to the registered queue.

The QNMRGTI API may be called to request the current topology only (without updates), updates only (without current topology), or both current topology and updates. This is determined by the options specified in the options parameter.

If an application program calling the API requests current topology, and the complete topology data cannot be returned in the user space, an error is returned. In this situation, the user space header contains a P in the information status field indicating partial but accurate data. Even if a user space error such as this occurs, any queue registered on the API call will remain registered.

Authorities and Locks

User Space Authority *CHANGE
User Space Library Authority *USE
Queue Authority *CHANGE
Queue Library Authority *USE
User Space Lock *EXCLRD

Required Parameter Group

Queue handle
OUTPUT; BINARY(4)
A variable that uniquely identifies the registered queue within the job. This value is returned when a queue is registered. When the *CURRENT value is the only option specified in the options parameter, 0 is returned.

Options
INPUT; ARRAY OF CHAR(10)
An array structure containing options specifying the topology information to be reported. These options only apply to topology updates. All deletions are reported regardless of the type of updating information you want. One or more of the following values must be specified:

***CURRENT**	Report the current copy of the entire topology database.
***LOCALEN**	Register for local topology updates pertaining to adjacent end nodes.[3]
***LOCALNN**	Register for local topology updates pertaining to adjacent network nodes.[4]
***LOCALVN**	Register for local topology updates pertaining to adjacent virtual nodes.[5]
***NETNN**	Register for network topology updates pertaining to network nodes.
***NETVN**	Register for network topology updates pertaining to virtual nodes.

Number of options
INPUT; BINARY(4)
The number of options specified in the options parameter. Valid values are 1 through 6.

Format
INPUT; CHAR(8)
The content and format of the topology information reported. The valid value is:

APPN0100	The basic APPN topology information format.

[3] Local topology consists of the local node, adjacent nodes, and links to adjacent nodes. See "Local and Network Topology Updates" on page 42-2 for more details.

[4] This option must be used if updates pertaining to adjacent network nodes in a disconnected subnetwork, such as network nodes having a different network ID, are required.

[5] An APPN virtual node represents a connection network (for example, an attached token-ring network). For more information, see *APPN Guide*.

See "APPN0100 Format" on page 42-16 for a description of this format.

Qualified user space name
INPUT; CHAR(20)
The user space that is to receive current topology information. This parameter is ignored when the *CURRENT value is not specified on the options parameter. The first 10 characters specify the user space name, and the last 10 characters specify the library name.

The following special values are supported for the library name:

***LIBL** The library list.
***CURLIB** The job's current library.

Qualified queue name
INPUT; CHAR(20)
The queue that is to receive requested topology information. This parameter is ignored when the *CURRENT value is the only option specified on the options parameter. The first 10 characters specify the queue object name, and the last 10 characters specify the library name.

The following special values are supported for the library name:

***LIBL** The library list.
***CURLIB** The job's current library.

When special values are used for the library name, the actual name will be substituted when the API is called. The actual name will be used in future references to the object when topology updates are enqueued.

The following considerations apply to the queue specified on the API call:

- The queue object must exist when this API is called.
- There is no restriction that prevents applications in separate jobs from registering the same queue object. Therefore, each application should ensure it registers a unique queue to prevent duplicate topology updates from being reported to the same queue. For example, specifying a queue in library QTEMP ensures the queue is not being used by other jobs on the system.
- The maximum entry length of the queue must be at least 128 bytes.
- The sequence of the queue must be *FIFO.
- Keyed queues are not supported.
- A data queue defined with FORCE(*YES) is allowed, but is discouraged due to degraded performance.
- A data queue defined with SENDERID is allowed, but the application is responsible for ensuring sufficient record length to handle the additional data on queue elements.

Queue type
INPUT; CHAR(10)
The type of queue object. This parameter is ignored when *CURRENT is the only value specified on the options parameter. Otherwise, one of the following values must be specified:

***DTAQ** Data queue
***USRQ** User queue

Replace registration
INPUT; CHAR(10)
Whether this registration should replace a previous registration that has the same qualified queue name and type. This parameter is ignored when *CURRENT is the only value specified on the options parameter.

***YES** This registration replaces any previous registration with the same qualified queue name and type. The options specified on this registration replace options from any previous registration.
***NO** This registration does not replace any existing registration with the same qualified queue name and type. The existing registration is not changed.

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 user space is used to report current topology information and 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 a detailed description of each field in the information returned, see "Field Descriptions" on page 42-17.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name specified
10	A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format name specified

Header Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library name used
20	14	BINARY(4)	Number of nodes reported
24	18	CHAR(8)	Timestamp

APPN0100 Format: The format of topology entry data is the same for queue entries reported on the registered queue for topology updates and deletions as for list entries reported to the user space for the current topology.

For topology entries reported to the user space, the entry ID field contains the value that indicates a current topology entry.

For topology entries reported to the registered queue, the entry ID field contains a value that identifies the type of notification reported:

- Topology entry updated
- Topology entry deleted
- Topology database deleted

The format of the topology entry is described below. See "Field Descriptions" on page 42-17 for descriptions of the fields in this format.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Entry type
10	A	CHAR(2)	Entry ID
12	C	CHAR(8)	Timestamp
20	14	CHAR(8)	Node network ID
28	1C	CHAR(8)	Node control point name
36	24	BINARY(4)	Number of associated TG entries
40	28	CHAR(1)	Node data valid indicator
41	29	CHAR(3)	Node type
44	2C	CHAR(8)	Node attributes
52	34	CHAR(8)	TG destination network ID
60	3C	CHAR(8)	TG control point name
68	44	BINARY(4)	TG number
72	48	CHAR(20)	TG characteristics
92	5C	BINARY(4)	Length of DLC signaling information
96	60	CHAR(16)	DLC signaling information
112	70	CHAR(10)	Controller description object name
122	7A	CHAR(1)	TG flags
123	7B	CHAR(5)	Reserved

Format of Node Attributes Field: The format of the node attributes field is described below.[6]

Byte	Bit	Field
0-3		Resource sequence number
4		Route addition resistance
5		Node status
	0	Node congested
	1	Intermediate routing resources depleted
	2	End point routing resources depleted
	3-4	Reserved
	5	Quiescing
	6-7	Reserved
6		Node type and support
	0-1	Reserved
	2	Currently in use
	3	Currently in use
	4-5	Reserved for management services
	6-7	Node type: **11** Node type 2.1

[6] See *System Network Architecture Formats* for details about SNA control vectors.

Byte	Bit	Field
7		Additional node support
	0	Adjacent subnetwork border node support: **0** The node lacks such support **1** The node has such support
	1-7	Reserved

Format of TG Flags Field: The structure of the TG flags data is described below.

Byte	Bit	Field
0	0	Link connection network indicator: **0** The TG-Partner Node's Network-Qualified CP/PU Name field does not identify a link connection network (such as a local area network) **1** The TG-Partner Node's Network-Qualified CP/PU Name field identifies a link connection network
	1-4	Reserved
	5	Intersubnetwork link indicator: **0** This link is not an intersubnetwork link. **1** This link is an intersubnetwork link. It defines a border between subnetworks.
	6-7	Reserved

Format of TG Characteristics: The format of the transmission group (TG) characteristics data is described below.[6]

Byte	Bit	Content
0-3		Resource sequence number (reserved except in topology database updates (TDUs)): a 32-bit binary value that uniquely identifies a topology update. See *System Network Architecture Formats*.
4		Status
	0	Operational status: **0** The TG is not operational **1** The TG is operational
	1	Reserved
	2	Quiescing: **0** The TG is not quiescing **1** The TG is quiescing
	3	CP-CP session support status: **0** CP-CP sessions are supported on this TG **1** CP-CP sessions are not supported on this TG
	4-7	Reserved
5		Effective capacity. See the *System Network Architecture Formats*.
6-10		Reserved
11		Cost per connect time
12		Cost per byte transmitted
13		Reserved
14		Security
15		Propagation delay of the TG, given as a floating-point number specifying microseconds
16		Reserved
17		User-defined parameter 1
18		User-defined parameter 2
19		User-defined parameter 3

Field Descriptions

Additional node support. Other services that may be supported.

Controller description object name. The name of the controller description object for the transmission group. This field is valid only when the number of associated TG entries field is not zero.

Cost per byte transmitted. The relative cost per byte of sending and receiving data on the line. Valid values are from 0 (lowest cost) to 255 (highest cost). This value is used for data flowing on the line after the initial connection is made.

Cost per connect time. The relative cost of being connected on the line. For example, connection to a non-switched line has the lowest relative cost and to a switched line a higher value. The cost for a nonswitched line is a set amount regardless of the amount of use. However, the cost for a switched line is based on the amount of time it was used.

DLC signaling information. The data link control (DLC) signaling information related to the link connection network. For token ring, the first 6 bytes is the MAC address, and the seventh byte is the link layer service access point address. This field is valid only when the number of associated TG entries field is not zero.

Effective capacity. A floating-point number, in units of 300 bits per second, representing the product of a user-defined maximum load factor and the bit transmission rate of the link underlying the TG.

Entry ID. The type of topology information. The possible values are listed below. Value 00 is always used in topology entries reported to only the user space. The other three values are used for topology entries reported to the registered queue.

00 The entry reported is present in the current APPN topology database.
01 The entry reported was updated in the APPN topology database.
02 The entry reported was deleted from the APPN topology database.[7]
03 The current APPN topology database was deleted[8] and reinitialized with the node entry for the local system. The node entry for the local system is reported on the entry. Any additions to the newly initialized topology database will follow in subsequent queue entries.

Entry type. The type of queue entry. This field can contain the following special value:

**APPNTOP* The entry contains APPN topology information.

Format name specified. The format name specified to the API.

Intersubnetwork link indicator. Defines an intersubnetwork link.

Length of DLC signaling information. The length of the data link control (DLC) signaling information. This field is valid only when the number of associated TG entries field is not zero. The value is 7 for token ring (only DLC currently allowed), or 0 when there is no DLC signaling information available.

Link connection network indicator. Defines a link to a connection between networks.

Node attributes. The attributes of the node. See "Format of Node Attributes Field" on page 42-16 for the structure. This field is valid only when the node data valid indicator field is Y. If the node data valid indicator field is N, the node attributes field contains data that is not valid.

Node control point name. The control point name for the node.

Node data valid indicator. Whether values contained in the node type and the node attributes fields are valid. The valid values are:

Y The values contained in the node type and the node attributes fields are valid.
N The values contained in the node type and the node attributes fields are not valid. This may occur when an entry is used to report only associated transmission group data links that originate from this node.

Node network ID. The network ID for the node.

Node status. The node type and data that may be reported as part of a topology update. A value of 1 indicates that a condition is true.

Node type. The APPN node type of the node. This value is blank when the node data valid indicator field is N. The valid values are:

**EN* APPN end node[9]
**NN* APPN network node
**VN* APPN virtual node

Node type and support. The node type and the services that may be supported.

Number of associated TG entries. The number of transmission group (TG) entries associated with updating or deleting this node. When this value is not zero, the specified number of associated TGs are reported for the node. The first associated TG entry is reported on the same entry as the initial node entry, and any additional TGs are reported on subsequent entries.

Number of nodes reported. The total number of node entries returned in the list (entries containing valid node data).

Propagation delay of the TG. This specifies the time required for a signal to travel from one end of a link to the other end. The delay is based on the protocol used and the physical connection. The value is given as a floating-point number specifying microsecond units.

Reserved. An ignored field.

Resource sequence number. A 32-bit binary value that uniquely identifies a topology update.

Route addition resistance. The ability of the node to accept additional routes. This field is a binary value in the range of 0 through 255. The greater the value, the less the ability of the node to accept additional routes.

Resource sequence number. A 32-bit binary value that uniquely identifies a topology update.

Security. The security used on the line. The security specified here is not related to the system user ID and password security. Possible values are:

**NONSECURE:* There is no security on the line.

---

7 APPN performs cleanup of the APPN topology database once every 24 hours. This cleanup may cause multiple entries to be removed from the APPN topology database.

8 This indication may be reported, for example, when the node type parameter changes as a result of the Change Network Attributes (CHGNETA) command, causing the APPN topology database to be deleted.

9 There is no distinction made between APPN end nodes and low-entry networking (LEN) nodes. Any node defined as a LEN node is reported as *EN.

*PKTSWTNET: This is a packet-switched network. Therefore, the line is secure in the sense that the data does not always use the same path through the network.
*UNDGRDCBL: This is an underground cable; secure conduit.
*SECURECND: This is a secured conduit but not guarded, such as pressurized pipe.
*GUARDCND: This line is a guarded conduit and protected against physical tapping.
*ENCRYPTED: Data flowing on the line is encrypted. This does not mean that the Operating System/400 licensed program is performing encryption. Separate hardware and programs can be purchased to perform encryption.
*MAX: This is a guarded conduit, protected against physical and radiation tapping.

Status. The status of the transmission group.

TG characteristics. The transmission group (TG) characteristics. See "Format of TG Characteristics" on page 42-17 for the structure. This field is valid only when the number of associated TG entries field is not zero.

TG control point name. The control point name for the transmission group (TG) destination node. This field is blank when the number of associated TG entries field is zero.

TG destination network ID. The network ID the for transmission group (TG) destination node. This field is blank when the number of associated TG entries field is zero.

TG flags. The format of the TG flags data is described in "Format of TG Flags Field" on page 42-17. This field is valid only when the number of associated TG entries field is not zero.

TG number. The transmission group (TG) number. This field is valid only when the number of associated TG entries field is not zero.

Timestamp. The machine timestamp (time of day) when reported.

User-defined parameter 1. The first of three user-definable fields. This field is used to describe any unique characteristics of the line that you want control over. The default value is 128.

User-defined parameter 2. The second of three user-definable fields. This field is used to describe any unique characteristics of the line that you want control over. The default value is 128.

User-defined parameter 3. The third of three user-definable fields. This field is used to describe any unique characteristics of the line that you want control over. The default value is 128.

User space library name specified. The user space library name specified to the API.

User space library name used. The actual user space library name used to report data.

User space name specified. The user space name specified to the API.

User space name used. The actual user space name used to report data.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C21 E Format name &1 is not valid.
CPF3CAA E List is too large for user space &1.
CPF3CF1 E Error code parameter not valid.
CPF8100 E All CPF81xx messages could be signaled. xx is from 01 to FF.
CPF91C2 E Queue type &1 not valid.
CPF91C3 E Internal processing error.
CPF91C4 E Queue &5/&4 with type &6 already registered.
CPF91C5 E Queue &3/&2 not valid.
CPF91C7 E Options list not valid.
CPF91C8 E Replace registration value &1 not valid.
CPF9800 E All CPF98xx messages could be signaled. xx is from 01 to FF.

Register Filter Notifications (QNMRGFN) API

Parameters

Required Parameter Group:

1	Qualified filter name	Input	Char(20)
2	Filter type	Input	Char(10)
3	Error code	I/O	Char(*)

The Register Filter Notifications (QNMRGFN) API registers a filter to send notifications for a specific event to a data queue. This data queue is defined in the *SNDDTAQ action entry field of the registered filter. These notification records are the output of this API. All CL filter commands can be performed against a registered filter.

A filter can only be registered once. If a registered filter is being registered again, an error is returned. No checking is done to see if the registered filter is the same as another filter, such as an active user filter.

A check to prevent duplicate notifications for a single event is made when the registered filters are processed. If a notification record for a registered filter is the same as another notification record, such as an active user filter, the registered filter is not processed.

If an error occurs in accessing a registered filter, or while enqueueing the notification record, the filter is automatically deregistered, and an informational message (CPI91D5) is sent to the system operator (QSYSOPR) message queue. This prevents the system from encountering the error again.

The registration for a filter remains active after the initial program load of a system. This ensures that a product receives all notifications. A product should register its filter either when the product is installed or when the product is started. A product should check to see that its filter is registered at its startup time to ensure that its filter was not automatically deregistered. You do not have to deregister a filter to change the filter.

Authorities and Locks

Registered Filter *USE
Registered Filter Library
*USE

Required Parameter Group

Qualified filter name
INPUT; CHAR(20)
The qualified filter name that is being registered. The first 10 characters are the filter name, and the last 10 characters are the library name. Special values *LIBL and *CURLIB are not supported for the library name.

For details about the format, see "Format of Registered Filter Data Queue Notification."

Filter type
INPUT; CHAR(10)
The type of filter. The following filter types are specified in the format field. Special values supported are:

***ALR** Alert filter. (See "Alert Filter"for details about the format.)
***PRB** Problem log filter. (See "Problem Log Filter" for details about the 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.

Format of Registered Filter Data Queue Notification

See "Field Descriptions" for descriptions of the fields in this format.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Notification name
10	A	CHAR(2)	Function code
12	C	CHAR(8)	Format
20	14	CHAR(10)	Filter library name
30	1E	CHAR(10)	Filter name
40	28	CHAR(10)	Group name
50	32	CHAR(8)	Timestamp
58	3A	CHAR(22)	Reserved
80	50	CHAR(*)	Event-specific notification data

The types of event-specific notification data used by the format field are described below.

Alert Filter: See "Field Descriptions" for descriptions of the fields in this format.

Offset			
Dec	Hex	Type	Field
80	50	CHAR(512)	Alert major vector

Problem Log Filter: See "Field Descriptions" for descriptions of the fields in this format.

Offset			
Dec	Hex	Type	Field
80	50	CHAR(30)	Problem ID
110	6E	CHAR(1)	Last event
111	6F	CHAR(8)	Last event timestamp

Field Descriptions

Alert major vector. The actual alert that caused the notification.

Event-specific notification data. Data specific to the event identified in the function code. The data is in the format specified by the format variable.

Filter library name. The library that contains the filter.

Filter name. The name of the filter that sent the notification.

Format. The format of the event-specific data of a problem log filter. Valid values are:

RGFN0100 Alert filter
RGFN0200 Problem log filter

Function code. The event that caused the notification. Valid values are:

01 Alert event
02 Problem log event

Group name. The group into which the event was filtered.

Last event. The last event performed on the problem.

Last event timestamp. The time at which the last event was performed.

Notification name. The type of notification record. It is set to *RGFN for all registered filter notifications.

Problem ID. The ID of the problem that caused the notification.

Reserved. An ignored field.

Timestamp. The time at which the event was processed.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C7C E User index is full.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF91D2 E Filter type &3 not correct for this operation.
CPF91D3 E Filter &1/&2 with type &3 is already registered.
CPF9800 E All CPF98xx messages could be signaled. xx is from 01 to FF.

Retrieve Alert (QALRTVA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	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

Receiver variable
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 receiver variable
INPUT; BINARY(4)
The length of the receiver variable. 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 42-22 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 42-22 for descriptions of the fields in this format.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	CHAR(8)	Timestamp
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)	Timestamp
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.

Timestamp. 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.
| 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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

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)
The unique identifier for this application, which was returned by the Start Application (QNMSTRAP) API.

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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Registered Filters (QNMRRGF) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Filter type	Input	Char(10)
4	Format	Input	Char(8)
5	Error code	I/O	Char(*)

The Retrieve Registered Filters (QNMRRGF) API returns all filters registered for a filter type. If not enough space exists to return all the registered filters, then as many registered filters as fit in the provided space are returned. A count of the total number of registered filters for the specific filter type is also returned.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable in which the registered filters are returned. If the variable is not large enough to contain all the registered filters, then only as many registered filters will be returned as will fit in the receiver variable.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable parameter. If not enough space is given, then only the number of filters that fit will be returned. In this case the number of bytes available will be greater than the number of bytes returned.

Filter type
INPUT; CHAR(10)
The type of filter to be retrieved. Special values supported are:

***ALR** Alert filter
***PRB** Problem log filter

Format
INPUT; CHAR(8)
The format of the receiver variable. The valid value is format RGFN0100. See "RGFN0100 Format" on page 42-24 for the format of the receiver variable.

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.

RGFN0100 Format

See "Field Descriptions" on page 42-24 for a description of the fields in this format.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Returned registered filters
12	C	BINARY(4)	Total registered filters
16	10	BINARY(4)	Element length
24	18	CHAR(*)	Array of registered filters
0	0	CHAR(10)	Filter name
10	A	CHAR(10)	Filter library
20	14	CHAR(10)	Filter type
Note: This is an array with the following element structure:			
0	0	CHAR(10)	Filter name
10	A	CHAR(10)	Filter library
20	14	CHAR(10)	Library type

Field Descriptions

Array of registered filters. A 10-character filter name followed by a 10-character library name followed by a 10-character filter type.

Bytes available. The number of bytes of data that is available. If this number is greater than the bytes returned, the receiver variable was not large enough to contain all the registered filters.

Bytes returned. The number of bytes of data returned in the receiver variable.

Element length. The length of a single array element. For format RGFN0100 the length is 30 bytes.

Filter library. The name of the library containing the registered filter.

Filter name. The name of the filter object that was registered.

Filter type. The type of the filter as defined by the FTRTYPE parameter on the Create Filter (CRTFTR) command.

Returned registered filters. The number of registered filters that is returned in the array.

Total registered filters. The total number of filters registered for the specified filter type.

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.
CPF3C24 E Length of the receiver variable is not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF91D6 E Filter type &1 not valid.
CPF9800 E All CPF98xx messages could be signaled. xx is from 01 to FF.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &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)

The unique identifier for this application, which was returned by the Start Application (QNMSTRAP) API.

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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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" for more information about sending a request.

Required Parameter Group

Handle
INPUT; BINARY(4)
The unique identifier for this application.

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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

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)
The unique identifier for this application, which was returned by the Start Application (QNMSTRAP) API.

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.
CPF7AE1 E Sign-on to control point &3.&4 denied.
CPF7AE2 E Handle &1 not found.
CPF7AE3 E Management Services transport operation not permitted.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

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.
| CPF9800 E All CPF98xx messages could be signaled. xx is from 01 to FF.

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	Binary(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(*)

Optional Parameter Group:

	11	Note text	Input	Char(80)
	12	Length of note text	Input	Binary(4)

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

Display panels
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.
***ANSWERED** The problem has been answered.

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.

Optional Parameter Group

Note text
INPUT; CHAR(80)
The field used to include a note stating that a problem originated with the service director. The note will be included when the API is called.

Length of note text
INPUT; BINARY(4)
The length of the text for the note text field. The text field for the note text field can be up to 80 characters long.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.

| CPF7AA7 E Problem &1 not found.
| CPF7A9C E Cannot work with the problem log at this time.
| CPF7A9D E Problem log object is missing.
| CPF7A93 E Problem &2 currently in use by job &1.
| CPF8C09 E &1 not defined as a service provider.
| CPF8C24 E Error occurred while processing request.
| CPF8C87 E Service provider &1.&2 not found.
| CPF9308 E Unable to complete problem analysis. Reason code &1.
| CPF931C E Problem analysis results not recorded in problem log.
| CPF9310 E Problem analysis procedure was exited before it had completed.
| CPF9313 E Requested procedure is not allowed.
| CPF932A E Number of requested statuses is not valid.
| CPF932B E *IBMSRV not allowed as service provider for problems in OPENED status.
| 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.
| 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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Program Number: 5738-SS1

Printed in Denmark by Bonde's

SC41-8223-02