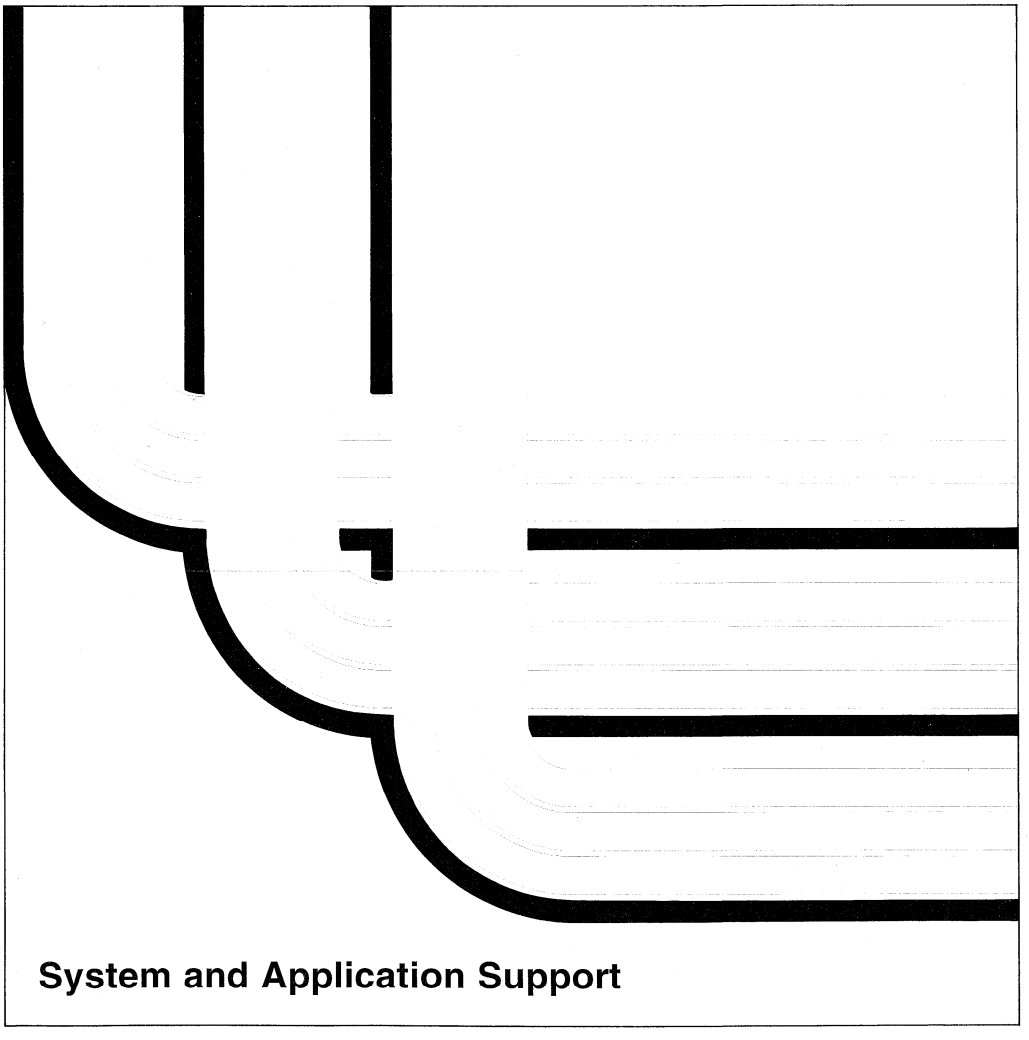
Application System/400

SC41-8223-02

System Programmer's Interface Reference Volume 2

Version 2



System and Application Support

Part 14. Object APIs

Chapter 43. User Space APIs

This chapter describes the user space APIs. You can use these APIs to create, change, delete, and retrieve information from user spaces.

User spaces are objects that consist of a collection of bytes used for storing user-defined information. They are permanent objects that are located in either the system domain or the user domain and have an object type of *USRSPC and a maximum size of 16MB. You can save and restore user spaces to other systems. However, if the user spaces contain pointers, you cannot restore the pointers even if you want to restore them to the same system.

You can use the user space APIs to:

- Allow list APIs to generate lists of data.
- Store data and pointers. There are some restrictions on storing pointers into user space objects in the system domain. If you are running at a security level of 40 or greater, you cannot directly manipulate a system-domain user space. Running at security level 40 or greater prevents you from storing pointers into a user space or retrieving pointers from a system-domain user space.
- Save and restore the data using CL commands.
- Create a user space as large as 16 megabytes. You cannot create a data area larger than 2000 bytes.
- Pass data from job to job or from system to system.

  Note: If the allow user domain (QALWUSRDMN) system value contains only the QTEMP library, this function is not possible. Refer to the *Guide to Enabling C2 Security* manual for more information.

The user space APIs include the following:

Change User Space (QUSCHGUS) changes the contents of a user space.

Change User Space Attributes (QUSCUSAT) changes the attributes of a user space object.

Create User Space (QUSCRTUS) creates a user space.

Delete User Space (QUSDLTUS) deletes user spaces created with the QUSCRTUS API.

Retrieve Pointer to User Space (QUSPTRUS) retrieves a pointer to the beginning of a user space for a high-level language (HLL) that supports pointers. HLLs that support pointers can use this pointer to manipulate the contents of a user space directly.

Retrieve User Space (QUSRTVUS) retrieves the contents of a user space. It does not receive descriptive information about the user space, such as its size.

Retrieve User Space Attributes (QUSRUSAT) retrieves information about creation attributes and current operational statistics of the user space.

The APIs in this chapter are presented in alphabetical order.

Change User Space (QUSCHGUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Starting position	Input	Binary(4)
3	Length of data	Input	Binary(4)
4	Input data	Input	Char(*)
5	Force changes to auxiliary storage	Input	Char(1)

Optional Parameter:

6	Error code	I/O	Char(*)

The Change User Space (QUSCHGUS) API changes the contents of the user space (*USRSPC) object by moving a specified amount of data to the object. This API allows you to change the contents of a user space if you are using either:

- A language that does not support pointers
- System-domain user spaces

Note: To determine the starting position for the QUSCHGUS API, you must add 1 to the offset value. In contrast to the OS/400 list APIs, which use an offset value based on 0 for the starting position, the QUSCHGUS API uses a value based on 1. For the QUSCHGUS API, the first character in the user space is at position 1.

Authorities and Locks

Library Authority *USE
User Space Authority *CHANGE
User Space Lock *EXCLRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The special values supported for the library name are *LIBL and *CURLIB.

Starting position
INPUT; BINARY(4)
The first byte of the user space that is to be changed. It must have a value greater than 0.

Length of data
INPUT; BINARY(4)
The length of the new data in the input data parameter. The length must be greater than 0.

Input data

INPUT: CHAR(*)

The new data to be placed into the user space. The field must be at least as long as the length of data parameter.

Force changes to auxiliary storage

INPUT; CHAR(1)

The method of forcing changes made to the user space to auxiliary storage. The valid values are as follows:

0 Does not force changes. Normal system management writes the changes to auxiliary storage.

1 Forces changes asynchronously. This interrupts the normal system management and ensures that the user space is written to auxiliary storage.

2 Forces changes synchronously. This interrupts the normal system management and ensures that the user space is written immediately to auxiliary storage.

Optional Parameter

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C0F E Value &1 for starting position parameter is not valid.
CPF3C04 E User space &1 not changed.
CPD3C0F D Value &1 for starting position parameter is not valid.
CPD3C12 D Value for length of data parameter is not valid.
CPD3C13 D Value &1 for force option is not valid.
CPD3C14 D Starting position &1 and length &2 cause space overflow.
CPD3C15 D New value &1 is shorter than the length specified.
CPD3C17 D Error occurred with source variable.
CPF3C12 E Length of data is not valid.
CPF3C13 E Value &1 for force option is not valid.
CPF3C14 E Starting position &1 and length &2 cause space overflow.
CPF3C15 E New value &1 is shorter than the length specified.
CPF3C17 E Error occurred with input data parameter.
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.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Change User Space Attributes (QUSCUSAT) API

Parameters

Required Parameter Group:

1	Returned library name	Output	Char(10)
2	Qualified user space name	Input	Char(20)
3	Attributes to change	Input	Char(*)
4	Error code	I/O	Char(*)

The Change User Space Attributes (QUSCUSAT) API changes the attributes of a user space object. This API can be used to:

- Extend or truncate a user space
- Mark or unmark the user space as automatically extendible by the system
- Change the initial value to which future extensions of the user space will be set

Authorities and Locks

Library Authority *USE
User Space Authority *CHANGE, *OBJMGT
User Space Lock *EXCL

Required Parameter Group

Returned library name

OUTPUT; CHAR(10)

The name of the library that contains the changed user space object. If the space attributes are successfully changed, the name of the library in which the user space was found is returned.

Qualified user space name

INPUT; CHAR(20)

The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The special values supported for the library name are *LIBL and *CURLIB.

Attributes to change
INPUT; CHAR(*)
The attributes of the user space object that you want to change. The information must be in the following format:

Number of attribute records
BINARY(4)
The total number of all of the attribute records.
Attribute records
The attributes of the user space to change and the data used for the change. Refer to "Format for Attribute Record" for more information.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format for Attribute Record

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Key
4	4	BINARY(4)	Length of data
8	8	CHAR(*)	Data

If you specify a length of data that is longer than the key field's defined data length, the data will be truncated at the right. No error message will be returned.

If you specify a length of data that is shorter than the key field's defined data length, an error message will be returned.

You may specify a key more than once. If duplicate keys are specified, the last specified value for that key is used.

Field Descriptions

Data. The value to which a specific user space attribute is to be changed. All values are validity checked.

Key. The user space attribute to be changed. Only specific attributes can be changed. Refer to "Keys" for more information.

Length of data. The length of the new user space attribute value. The length of data field is used to get the addressability to the next attribute record.

Keys: The following table lists the keys that can be used in the attribute record.

Key	Type	Attribute
1	BINARY(4)	Space size
2	CHAR(1)	Initial value
3	CHAR(1)	Automatic extendibility

Field Descriptions

Automatic extendibility. Whether or not the user space is automatically extended by the system when the end of the space is encountered.

0 The user space is not automatically extendible.
1 The user space is automatically extendible.

Initial value. The initial value to which future extensions of the user space will be set to. You will achieve the best performance if you set this byte to hexadecimal zeros (X'00').

Space size. The size in bytes of the user space object. If this value is smaller than the current size of the space, the user space is truncated. If it is larger, the space is extended.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C4B E Value not valid for field &1.
CPF3C4C E Value not valid for field &1.
CPF3C4D E Length &1 for key &2 not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
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.

Create User Space (QUSCRTUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Extended attribute	Input	Char(10)
3	Initial size	Input	Binary(4)
4	Initial value	Input	Char(1)
5	Public authority	Input	Char(10)
6	Text description	Input	Char(50)

Optional Parameter Group 1:

7	Replace	Input	Char(10)
8	Error code	I/O	Char(*)

Optional Parameter Group 2:

9	Domain	Input	Char(10)

The Create User Space (QUSCRTUS) API creates a user space in either the user domain or the system domain. A system-domain user space cannot be saved to a release prior to Version 2 Release 3 Modification 0. A user-domain user space can be directly manipulated with machine interface (MI) instructions or can be accessed using system APIs. On systems with a QSECURITY system value of 40 or greater, applications can only access system-domain user spaces using APIs. The user space objects you create are larger than or equal to the size specified. They have a fixed length and can be extended or truncated using the Change User Space Attributes (QUSCUSAT) API or the Modify Space (MODS) MI instruction (for user-domain user spaces). (The MODS instruction will not work on system-domain user spaces if the security level of the system is 40 or greater.)

Authorities and Locks

User Space Authority
*OBJMGT, *OBJEXIST, and *READ. These authorities are required only if the replace parameter is used and if there is an existing user space to replace.

User Space Library Authority
*USE and *ADD. *OBJOPR plus *READ is equivalent to *USE.

User Space Lock
*EXCL. This applies to both the user space being created and an existing user space being replaced.

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The only special value supported for the library name is *CURLIB.

User spaces created in the QTEMP and QRPLOBJ libraries are not forced to permanent storage; they are deleted when those libraries are cleared at sign-off and system IPL, respectively.

Extended attribute
INPUT; CHAR(10)
The extended attribute of the user space. For example, an object type of *FILE has an extended attribute of PF (physical file), LF (logical file), DSPF (display file), SAVF (save file), and so on.

The extended attribute must be a valid *NAME. You can enter this parameter in uppercase, lowercase, or mixed case. The API converts it to uppercase.

Initial size
INPUT; BINARY(4)
The initial size of the user space being created. This value must be from 1 byte to 16 776 704 bytes.

Initial value
INPUT; CHAR(1)
The initial value of all bytes in the user space. You will achieve the best performance if you set this byte to X'00'.

Public authority
INPUT; CHAR(10)
The authority you give users who do not have specific private or group authority to the user space. Once the user space has been created, its public authority stays the same when it is moved to another library or restored from backup media.

If the replace parameter is used and a user space exists to be replaced, this parameter is ignored. All authorities are transferred from the replaced user space to the new one.

The valid values for this parameter are:

***ALL**
The user can perform all authorized operations on the object.

Authorization list name
The user space is secured by the specified authorization list, and its public authority is set to *AUTL. The specified authorization list must exist on the system when this API is called. If it does not exist, the create process fails, and an error is returned to the application.

***CHANGE**
The user can read the object description and has read, add, update, and delete authority to the object.

***EXCLUDE**
The user cannot access the object in any way.

***LIBCRTAUT**
The public authority for the user space is taken from the CRTAUT value for the target library when the object is created. If the CRTAUT value for the library changes later, that change does not affect user spaces already created. If the CRTAUT value contains an authorization list name and that authorization list secures an object, do not delete the list. If you do, the next time you call this API with the *LIBCRTAUT parameter, it will fail.

***USE**
The user can read the object and its description but cannot change them.

Text description
INPUT; CHAR(50)
This text briefly describes the user space.

Optional Parameter Group 1

Replace
INPUT; CHAR(10)
Whether you want to replace an existing user space.

Valid values for this parameter are:

***NO** Do not replace an existing user space of the same name and library. *NO is the default value.

***YES** Replace an existing user space of the same name and library.

If the user space already exists, it is replaced by a new user space of the same name and library, and is subject to the same authorities. The user space being replaced is destroyed if both:

- The allow user domain (QALWUSRDMN) system value is not set to *ALL or does not contain the library QRPLOBJ.
- The user space you are replacing is in the user domain.

If the user space is in the system domain, it is moved to QRPLOBJ. If QALWUSRDMN is set to *ALL or if it contains QRPLOBJ, the replaced user space is moved to QRPLOBJ, which is cleared at system IPL. For details about authorities, ownership, and renaming, see the discussion of the REPLACE parameter in Volume 3 of 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. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Optional Parameter Group 2

Domain
INPUT; CHAR(10)
The domain into which the user space is created. If this parameter is not specified, the value of *DEFAULT is assumed by the API. Valid values for this parameter are:

***DEFAULT** Allows the system to decide into which domain the object should be created.
***SYSTEM** Creates the user space object into the system domain. The API can always create a user space into the system domain regardless of the security level in effect. However, you must use APIs to access system-domain user spaces if you are running at security level 40 or greater.
***USER** Attempts to create the user space object into the user domain. This is not always possible. If the library you are creating the user space into does not appear in the QALWUSRDMN system value, the API cannot create the user space into the user domain. An error will be returned.

The API uses the following values to determine into which domain to create the user space. The destination library is the library you specified in the qualified user space name parameter. The optional domain parameter is the information specified in the domain parameter.

QALWUSRDMN System Value	Destination Library	Optional Domain Parameter	Domain of Created Object
*ALL	Any	*DEFAULT	User domain (no change)
*ALL	Any	*SYSTEM	System domain
*ALL	Any	*USER	User domain
QTEMP	QTEMP	*DEFAULT	User domain
QTEMP	QTEMP	*SYSTEM	System domain
QTEMP	QTEMP	*USER	User domain
Does not contain library name	Library name	*DEFAULT	System domain
Does not contain library name	Library name	*SYSTEM	System domain
Does not contain library name	Library name	*USER	None; error is returned

Note: The QALWUSRDMN system value lists the libraries into which user domain objects can be created. The libraries can be the special value *ALL or a list of one or more library names.

You must use APIs to access data or information in system-domain user spaces on systems with a QSECURITY level of 40 or greater. You cannot use MI instructions to directly access system-domain user objects.

The Retrieve Object Description (QUSROBJD) or List Objects (QUSLOBJ) API can be used to determine into which domain the user-space object was created.

Error Messages

CPF2143 E Cannot allocate object &1 in &2 type *&3.
CPF2144 E Not authorized to &1 in &2 type *&3.
CPF2283 E Authorization list &1 does not exist.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C01 E User space &2 in library &1 not created.
CPD3C01 D Object name &1 is not valid.
CPD3C03 D Extended attribute &1 is not valid.
CPD3C04 D Value &1 for size parameter is not valid.
CPD3C05 D Value &1 for authority parameter is not valid.
CPF3C2B E Extended attribute &1 is not valid.
CPF3C2C E Value &1 for size parameter is not valid.
CPF3C2D E Value &1 for authority parameter is not valid.
CPF3C29 E Object name &1 is not valid.
CPF3C34 E Value &1 for replace option is not valid.

CPF3C36 E Number of parameters, &1, entered for this API was not valid.
| CPF3C45 E Value &1 not valid for domain parameter.
| CPF3C49 E Request for user domain object cannot be granted.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.
CPF9870 E Object &2 type *&5 already exists in library &3.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Delete User Space (QUSDLTUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Error code	I/O	Char(*)

The Delete User Space (QUSDLTUS) API deletes user spaces created with the Create User Space (QUSCRTUS) | API. The QUSDLTUS API performs the same function as | the Delete User Space (DLTUSRSPC) command.

Authorities and Locks

Library Authority *READ
User Space Authority *OBJEXIST and *USE
User Space Lock *EXCL

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the user space and the name of the library in which it resides. The first 10 characters contain the user space name, and the second 10 characters contain the library name.

The user space name can be either a specific name or a generic name, a string of one or more characters followed by an asterisk (*). If you specify a generic name, QUSDLTUS deletes all user spaces that have names beginning with the string for which the user has authority.

You can use these special values for the library name:

***ALL** All libraries
***ALLUSR** All user-defined libraries, plus libraries containing user data and having names starting with Q
***CURLIB** The job's current library
***LIBL** The library list
***USRLIBL** The user portion of the job's library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF2105 E Object &1 in &2 type *&3 not found.
CPF2110 E Library &1 not found.
CPF2113 E Cannot allocate library &1.
CPF2114 E Cannot allocate object &1 in &2 type *&3.
CPF2117 E &4 objects type *&3 deleted. &5 objects not deleted.
CPF2125 E No objects deleted.
CPF2176 E Library &1 damaged.
CPF2182 E Not authorized to library &1.
CPF2189 E Not authorized to object &1 in &2 type *&3.
| CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Pointer to User Space (QUSPTRUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Return pointer	Output	PTR(SPP)

Optional Parameter:

3	Error code	I/O	Char(*)

The Retrieve Pointer to User Space (QUSPTRUS) API | retrieves a pointer to the contents of a user-domain user | space. The data in that user space then can be directly | manipulated by high-level language programs that support | pointers. (PL/I, Pascal, C/400, COBOL/400, and System | C/400 PRPQ support pointers.) The QUSPTRUS API will | not return a pointer to a system-domain user space; you | must use system APIs to access system-domain user | spaces. If you attempt to retrieve the pointer of a system-| domain user space, an error will be returned.

| The QUSPTRUS API even returns a pointer to an object that is subject to an exclusive (*EXCL) lock. If you create application programs using HLLs that can directly update user spaces using pointers (instead of using the Change User Space (QUSCHGUS) API), you should use your own synchronization data methods using the LOCK or LOCKSL machine interface (MI) instruction or the Allocate Object (ALCOBJ) command to avoid updates at the same time to the same location within a user space.

Use of the QUSPTRUS API does not update the object usage information (such as last changed date, last date used, and so on). You should use the Change User Space or the Retrieve User Space API to update the object usage information if needed.

Authorities and Locks

Library Authority *USE
User Space Authority *USE

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The special values supported for the library name are *LIBL and *CURLIB.

Return pointer
OUTPUT; PTR(SPP)
The variable containing the pointer to the user space after the QUSPTRUS API has completed running. This parameter must be on a 16-byte boundary alignment.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C05 E One or more errors found while trying to retrieve a pointer.
CPF3C18 E Pointer parameter is not on a 16-byte boundary.
CPD3C18 D Pointer parameter is not on a 16-byte boundary.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF3C48 E Operation not valid on system domain object.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Example: Retrieving a Pointer with a Pascal Program

To retrieve the pointer to a user space with a call from a Pascal program, specify the following:

```
(*********************************************************************)
(***                                                               ***)
(*** PROGRAM:  POINTER                                            ***)
(***                                                               ***)
(*** LANGUAGE:  PASCAL                                            ***)
(***                                                               ***)
(*** DESCRIPTION:  USE A POINTER TO RETRIEVE CONTENTS OF A USER  ***)
(***              SPACE.                                           ***)
(***                                                               ***)
(*** APIs USED:  QUSPTRUS                                         ***)
(***                                                               ***)
(*********************************************************************)
program Pointer;

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

    theInformation = record
                       field1: packed array(.1..100.) of char;
                       field2: Integer;
                       field3: Integer;
                     end;
    InfoPtrType    = @theInformation;

  var
    theUserSpace           : UserSpaceName;
    thePtr                 : InfoPtrType;

    ScratchVar             : Integer;

  procedure QUSPTRUS (var UserSpace: UserSpaceName;
                      var PtrToInfo : InfoPtrType);
                      nonpascal;

begin
  theUserSpace := 'TEMPSPACE QGPL';

  QUSPTRUS(theUserSpace,thePtr);

  ScratchVar:=thePtr@.Field2;

end.
```

Additional examples of the API are in Appendix A, "Examples."

Retrieve User Space (QUSRTVUS) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Starting position	Input	Binary(4)
3	Length of data	Input	Binary(4)
4	Receiver variable	Output	Char(*)

Optional Parameter:

5	Error code	I/O	Char(*)

The Retrieve User Space (QUSRTVUS) API allows you to retrieve the contents of a user space. The QUSRTVUS API does not retrieve descriptive information about the user

space object, such as its size. To retrieve information about the attributes of a user space, use one of the following:

- The Materialize Space (MATS) machine interface (MI) instruction
- The Retrieve Object Description (QUSROBJD) API described on page 46-12
- The Retrieve Object Description (RTVOBJD) command
- The Retrieve User Space Attributes (QUSRUSAT) API

If you are repeatedly accessing the contents of a user space and are using an HLL that supports pointers, see "Retrieve Pointer to User Space (QUSPTRUS) API" on page 43-6; this API provides a pointer to the user space for improved performance. When you have obtained a pointer, you use pointer arithmetic to access the contents of a user space.

Note: To determine the starting position for the QUSRTVUS API, you must add 1 to the offset value. In contrast to the OS/400 list APIs, which use an offset value based on 0 for the starting position, the QUSRTVUS API uses a value based on 1. For the QUSRTVUS API, the first character in the user space is at position 1.

Authorities and Locks

Library Authority *USE
User Space Authority *USE
User Space Lock *SHRNUP

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The special values supported for the library name are *LIBL and *CURLIB.

Starting position
INPUT; BINARY(4)
The first byte of the user space to be retrieved. A value of 1 will identify the first character in the user space.

Length of data
INPUT; BINARY(4)
The length of the data to retrieve. This length must not be larger than the size of the variable that is to receive the data. It must also be greater than 0.

Receiver variable
OUTPUT; CHAR(*)
The variable that will receive the contents of the user space being retrieved.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C0F E Value &1 for starting position parameter is not valid.
CPF3C06 E Information not retrieved from user space &1.
CPD3C0F D Value &1 for starting position parameter is not valid.
CPD3C12 D Value for length of data parameter is not valid.
CPD3C14 D Starting position &1 and length &2 cause space overflow.
CPD3C16 D Receiver area too small for length of data &1 specified.
CPD3C20 D Error occurred with receiver variable specified.
CPF3C12 E Length of data is not valid.
CPF3C14 E Starting position &1 and length &2 cause space overflow.
CPF3C16 E Receiver area too small for length of data &1 specified.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve User Space Attributes (QUSRUSAT) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified user space name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve User Space Attributes (QUSRUSAT) API retrieves information about the current attributes and the current operational statistics of the user space.

Authorities and Locks

User Space Library Authority
*USE

User Space Authority
*USE

User Space Lock *SHRNUP

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. You can specify the size of this area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the space information to be returned. The format names supported are:

SPCA0100 Basic information

Refer to "SPCA0100 Format" for details on the format.

Qualified user space name
INPUT; CHAR(20)
The user space for which you want to retrieve 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

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

SPCA0100 Format

The following information about a user space is returned for the SPCA0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Space size
12	C	CHAR(1)	Automatic extendibility
13	D	CHAR(1)	Initial value
14	E	CHAR(10)	User space library name

Field Descriptions

Automatic extendibility. Whether or not the space is extended automatically by the system when the end of the space is encountered.

0 Space is not automatically extendible
1 Space is automatically extendible

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.

Initial value. The initial value to which future extensions of the user space will be set.

Space size. The size of the user space object in bytes.

User space library name. The library in which the user space is located. This is helpful when *LIBL or *CURLIB is specified as the library name in the qualified user space name parameter.

Error Messages

CPF24B4 E Severe error 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 not valid.
CPF8100 E All CPF81xx messages could be signaled. xx is from 01 to FF.
CPF9801 E Object &1 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 libs in *LIBL previously deleted.
CPF9808 E Unable to allocate one or more libraries in *LIBL.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Unable to allocate library
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Chapter 44. User Index APIs

This chapter describes the user index APIs, which allow you to:

- Create and delete user indexes
- Add, retrieve, and remove user index entries
- Retrieve the attributes of a user index

A **user index** is an object that allows search functions and automatically sorts data based on the value of the data. User indexes are permanent objects in the user domain or in the system domain. They have an object type of *USRIDX and a maximum size of 4 gigabytes (4 294 967 296 bytes). They help streamline table searching, cross-referencing, and ordering of data. In general, if your table is longer than 1000 entries, an index performs faster than a user-sorted table.

You can use user indexes to:

- Provide search functions
- Do faster insert operations than in a database file
- Do faster retrieve operations than in a database file
- Create an index by name, such as a telephone directory
- Use order entry programs
- Look up abbreviations in an index
- Sort data automatically

For more information about user index considerations, refer to "User Index Considerations" on page 2-11. User index entries cannot contain a pointer. You can save and restore all the data in an index. You can also save and restore user indexes to another system.

The user index APIs are:

Add User Index Entries (QUSADDUI) allows you to add one or more new entries into the user index.
Create User Index (QUSCRTUI) creates a user index.
Delete User Index (QUSDLTUI) deletes a user index.
Remove User Index Entries (QUSRMVUI) removes one or more user index entries that match the remove criteria specified.
Retrieve User Index Attributes (QUSRUIAT) retrieves information about the user index attributes, including when it was created. It also retrieves the current operational statistics of the user index.
Retrieve User Index Entries (QUSRTVUI) retrieves user index entries that match the search criteria specified.

You can also work with user indexes through the System C/400 PRPQ or machine interface (MI) instructions and the Delete User Index (DLTUSRIDX) command. See the *Languages: System C/400 Programming RPQ P10102 User's Guide and Reference* or the *MI Functional Reference* for details about MI instructions and the *CL Reference* for details about the DLTUSRIDX command.

Add User Index Entries (QUSADDUI) API

Parameters

Required Parameter Group:

1	Returned library name	Output	Char(10)
2	Number of entries added	Output	Binary(4)
3	Qualified user index name	Input	Char(20)
4	Insert type	Input	Binary(4)
5	Index entries	Input	Char(*)
6	Length of index entries	Input	Binary(4)
7	Entry lengths and offsets	Input	Array(*) of Char(8)
8	Number of entries	Input	Binary(4)
9	Error code	I/O	Char(*)

The Add User Index Entries (QUSADDUI) API inserts one or more new entries into the user index by the insert type. Each entry is inserted into the index at the appropriate location based on the binary value of the entry. No other collating sequence is supported. Every entry added causes the number of entries added parameter to be incremented by 1; you can retrieve this information by using the Retrieve User Index Attributes (QUSRUIAT) API.

When you request to add multiple entries to a user index, the request may be partially successful in the following situations:

- One or more of the entries you requested to add already exists in the index.
- The index becomes full as the entries are added.
- The user profile storage limit is exceeded as entries are added.
- One or more of the entry lengths or offsets are not valid.

When an error occurs, you should check the number of entries added parameter to see if all entries were successfully added. If the number of entries added parameter and the number of entries parameter are not equal, then all entries were not added.

Note: If you add new entries with an entry length longer than the user index is expecting, the entries are truncated to the right. No error is given.

Authorities and Locks

User Index Library Authority
*USE
User Index Authority
*CHANGE (*OBJMGT, *OBJEXIST, *USE)
User Index Lock *EXCL

Required Parameter Group

Returned library name
OUTPUT; CHAR(10)
The name of the library that contains the user index to

which the entries were added if they were added successfully. This parameter is not set if no entries were successfully added. This information helps you identify the specific library used when *LIBL or *CURLIB is specified in the qualified user index name parameter.

Number of entries added
OUTPUT; BINARY(4)
The number of entries successfully added to the specified user index. If an error is received while processing the entries, this number indicates how many were added before the error occurred.

Qualified user index name
INPUT; CHAR(20)
The user index to which you want to add entries and the library in which it is located. The first 10 characters contain the user index 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

Insert type
INPUT; BINARY(4)
The type of insert to be performed against all entries that are to be added. Valid values are:

1 Insert unique argument
This value is only valid for nonkeyed user indexes. If the entry is already in the index or the index entries are keyed, an error is returned.
2 Insert with replacement
This value requests to replace the nonkey portion of the index entry if the key is already in the user index. It is only valid for keyed user indexes. If the entry does not exist, it will be inserted into the user index. If the index entries are nonkeyed, an error is returned.
3 Insert without replacement
This value requests to insert the entry only if the key is not already in the user index. It is only valid for a keyed user index. If the entry does not exist, it will be inserted into the user index. If the entry is already in the index or the index entries are nonkeyed, an error is returned.

Index entries
INPUT; CHAR(*)
The actual entry or entries to be added to the user index. If the user index contains *fixed length* entries, this parameter is processed using the entry length specified when the user index was created. If the user index contains *variable length* entries, this parameter is processed using the information contained in the entry lengths and entry offsets parameter.

Length of index entries
INPUT; BINARY(4)
The length of the index entries parameter. This value must be greater than 0.

Entry lengths and entry offsets
INPUT; ARRAY(*) of CHAR(8)
If the user index contains variable length entries, this parameter is a data-structure array that is used to parse through the index entries parameter. In this case, an entry length and entry offset need to be provided for all entries that are to be added. This parameter is ignored for user indexes containing fixed length entries.

The size of the entry lengths and entry offsets parameter must be at least eight times the number of entries parameter; otherwise, an error will be returned.

See "Format for Entry Lengths and Entry Offsets" for details on the data structure.

Number of entries
INPUT; BINARY(4)
The number of entries that are to be added to the user index. Valid values are 1 through 4095.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format for Entry Lengths and Entry Offsets

The following table defines the format for the entry lengths and entry offsets parameter. This information is needed to parse through the index entries parameter. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
Note: The following fields will be repeated. The number of times they are repeated depends on the value specified in the number of entries parameter.			
		BINARY(4)	Entry length
		BINARY(4)	Entry offset

Field Descriptions

Entry length. The length of the entry to be inserted into the index. Valid values are 1–2000. This value depends on how the user index was created.

Entry offset. For the first entry, the offset is the number of bytes from the beginning of the index entries parameter to the first byte of the first entry. For each subsequent entry, the offset is the number of bytes from the beginning of the previous entry to the first byte of the next entry. Each entry offset value must be greater than or equal to 0 and must refer to an entry within the index entries parameter.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C7B E Length of index entries is not valid.
CPF3C7C E User index is full.
CPF3C71 E Insert type &1 is not valid.
CPF3C72 E Number of entries &1 is not valid.
CPF3C73 E Error occurred with index entries parameter.
CPF3C74 E Entry is already in the index.
CPF3C75 E Error occurred with entry lengths and entry offsets parameter.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
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.

Create User Index (QUSCRTUI) API

Parameters

Required Parameter Group:

1	Qualified user index name	Input	Char(20)
2	Extended attribute	Input	Char(10)
3	Entry length attribute	Input	Char(1)
4	Entry length	Input	Binary(4)
5	Key insertion	Input	Char(1)
6	Key length	Input	Binary(4)
7	Immediate update	Input	Char(1)
8	Optimization	Input	Char(1)
9	Public authority	Input	Char(10)
10	Text description	Input	Char(50)

Optional Parameter Group 1:

11	Replace	Input	Char(10)
12	Error code	I/O	Char(*)

Optional Parameter Group 2:

13	Domain	Input	Char(10)

The Create User Index (QUSCRTUI) API creates a user index in either the user domain or the system domain. A system-domain user index cannot be saved to a release prior to Version 2 Release 3 Modification 0. A user-domain user index can be directly manipulated with MI instructions and can also be accessed using system APIs at all security levels. On a system with the QSECURITY system value set to 40 or greater, you must use system APIs to access system-domain user indexes.

If the user index was created prior to Version 2 Release 2 Modification 0, the size of the user index is limited to a maximum of 1 gigabyte. (A user index with a size greater than 1 gigabyte cannot be saved to or restored from a release prior to Version 2 Release 2 Modification 0.) If the user index was created on or after Version 2 Release 2 Modification 0, the size of the object is limited to a maximum of 4 gigabytes. The size is dependent on the amount of storage needed for the number and size of index entries and excludes the size of the associated space, if any.

Note: You can tell whether a user index object can be saved to a release prior to Version 2 Release 2 Modification 0:

- By ensuring that the current object size is less than 1 gigabyte by using one of the following:
  - The Display Object Description (DSPOBJD) command
  - The List Objects (QUSLOBJ) API
  - The Retrieve Object Description (QUSROBJD) API
- By ensuring that the key length field is 120 bytes or less by using either of the following:
  - The materialize index attributes (MATINXAT) MI instruction
  - The Retrieve User Index Attributes (QUSRUIAT) API

Authorities and Locks

Library Authority
*OBJOPR, *ADD, and *READ.

User Index Authority
*OBJMGT, *OBJEXIST, and *READ. These authorities are required only if the replace parameter is used and there is an existing user index to replace.

User Index Lock
*EXCL. This applies to both the user index being created and an existing user index being replaced.

Required Parameter Group

Qualified user index name
INPUT; CHAR(20)
The name of the user index being created, and the library in which it is to be located. The first 10 characters contain the user index name, and the second 10 characters contain the library name. You can use this special value for the library name:

***CURLIB** The job's current library

User indexes created in the QTEMP and QRPLOBJ libraries are not forced to permanent storage; they are deleted when those libraries are cleared at sign-off and system IPL, respectively.

Extended attribute
INPUT; CHAR(10)
The extended attribute of the user index. For example, an object type of *FILE could have an extended attribute of PF (physical file), LF (logical file), DSPF (display file), or SAVF (save file).

The extended attribute must be a valid *NAME. You can enter this parameter in uppercase, lowercase, or mixed case. The API automatically converts it to uppercase.

Entry length attribute
INPUT; CHAR(1)
Whether there are fixed-length or variable-length entries in the user index. The valid values are:

F Fixed-length entries
V Variable-length entries

Entry length
INPUT; BINARY(4)
The length of entries in the index entry.

The valid values for fixed-length entries are from 1 through 2000.

Valid values for variable length entries are 0 or -1. A value of 0 enables a maximum entry length of 120 bytes and a key length from 1 through 120. A value of -1 enables a maximum entry length of 2000 and a key length from 1 through 2000.

Note: A user index created with an entry length greater than 120 cannot be saved or restored to a release prior to Version 2 Release 2 Modification 0.

Key insertion
INPUT; CHAR(1)
Whether or not the inserts to the index are by key. The valid values are:

0 No insertion by key
1 Insertion by key

Key length
INPUT; BINARY(4)
The length of the key where the first byte of an entry is the beginning of the key for the index entries. The value for this parameter must be 0 for no insertion by key. If you specify key length insertion, this value is from 1 through 2000.

Immediate update
INPUT; CHAR(1)
Whether or not the updates to the index are written synchronously to auxiliary storage on each update to the index. The valid values are:

0 No immediate update
1 Immediate update

Each update to the index is written to auxiliary storage after every insert and remove operation.

Optimization
INPUT; CHAR(1)
The type of access in which to optimize the index. The valid values are:

0 Optimize for random references
1 Optimize for sequential references

Public authority
INPUT; CHAR(10)
The authority you give to users who do not have specific private or group authority to the user index. Once the user index has been created, its public authority stays the same when it is moved to another library or restored from backup media.

If the replace parameter is used and an existing user index is replaced, this parameter is ignored. All authorities are transferred from the replaced user index to the new one.

The valid values for this parameter are:

***ALL** The user can perform all authorized operations on the user index.
Authorization list name
The user index is secured by the specified authorization list, and its public authority is set to *AUTL. The specified authorization list must exist on the system when this API is issued. If the list does not exist, the create process fails, and an error message is returned to the application.
***CHANGE**
The user has read, add, update, and delete authority for the user index and can read the object description.
***EXCLUDE**
The user cannot access the user index in any way.
***LIBCRTAUT**
The public authority for the user index is taken from the CRTAUT value for the target library when the object is created. If the CRTAUT value for the library changes later, that change does not affect user indexes already created. If the CRTAUT value contains an authorization list name and that authorization list secures an object, do not delete the list. If you do, the next time you call this API with the *LIBCRTAUT parameter, it will fail.
***USE** The user can read the object description and contents but cannot change the user index.

Text description
INPUT; CHAR(50)
A brief description of the user index.

Optional Parameter Group 1

Replace
INPUT; CHAR(10)
Whether you want to replace an existing user index. Valid values for this parameter are:

***NO** Do not replace an existing user index of the same name and library. This is the default value.
***YES** Replace an existing user index of the same name and library.

If the user index already exists, you can replace it with a new user index of the same name and library. The new user index is subject to the same authorities. The user index being replaced is destroyed if both:

- The allow user domain (QALWUSRDMN) system value is not set to *ALL or does not contain the QRPLOBJ library.
- The user index you are replacing is in the user domain.

If the user index is in the system domain, it is moved to the QRPLOBJ library. If QALWUSRDMN is set to *ALL or if it contains QRPLOBJ, the replaced user index is moved to QRPLOBJ, which is cleared at system IPL. For details about authorities, ownership, and renaming, see the discussion of the REPLACE parameter in the appendix in Volume 3 of 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. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Optional Parameter Group 2

Domain
INPUT; CHAR(10)
The domain into which the user index should be created. If this parameter is not specified, the value of *DEFAULT will be assumed by the API. Valid values for this parameter are:

***DEFAULT**
Allows the system to decide into which domain the object should be created.
***SYSTEM** Creates the user index object into the system domain. The API can always create a user index into the system domain, regardless of the security level running. However, if you are running at security level 40 or greater, you must use APIs to access system-domain user index objects.
***USER** Attempts to create the user index object into the user domain. This is not always possible. If the library you are creating the user index into does not appear in the QALWUSRDMN system value, the API cannot create the user index into the user domain. An error message will be returned.

The API uses the following criteria to determine into which domain to create the user index. The destination library is the library you specified in the qualified user index name parameter. The optional domain parameter is the information specified in the domain parameter.

QALWUSRDMN System Value	Destination Library	Optional Domain Parameter	Domain of Created Object
*ALL	Any	*DEFAULT	User domain (no change)
*ALL	Any	*SYSTEM	System domain
*ALL	Any	*USER	User domain
QTEMP	QTEMP	*DEFAULT	User domain
QTEMP	QTEMP	*SYSTEM	System domain
QTEMP	QTEMP	*USER	User domain
Does not contain library name	Library name	*DEFAULT	System domain
Does not contain library name	Library name	*SYSTEM	System domain
Does not contain library name	Library name	*USER	None; error is returned

Note: The QALWUSRDMN system value lists the libraries into which the user domain objects can be created. Valid libraries are the special value *ALL or a list of one or more library names.

If your system is at security level 40 or greater, you must use APIs to access system-domain user indexes using APIs. You cannot use MI instructions to directly access system-domain user indexes.

You can use the Retrieve Object Description (QUSROBJD) API or the List Object (QUSLOBJ) API to determine into which domain the user index object was created.

Dependencies between Parameters

Some of the parameters are interdependent and are shown in the following table:

Entry Length Attribute	Entry Length (n)	Key Insertion	Key Length (x)
Fixed	$1 \le n \le 2000$	No	0

Entry Length Attribute	Entry Length (n)	Key Insertion	Key Length (x)
Fixed	1 ≤ n ≤ 2000	Yes	1 ≤ x ≤ n
Note: For the following entry lengths: • 0 signifies a maximum entry length of 120. • -1 signifies a maximum entry length of 2000.			
Variable	0, -1	No	0
Variable	0	Yes	1 ≤ x ≤ 120
Variable	-1	Yes	1 ≤ x ≤ 2000

Error Messages

CPF2143 E Cannot allocate object &1 in &2 type *&3.
CPF2144 E Not authorized to &1 in &2 type *&3.
CPF2283 E Authorization list &1 does not exist.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C0A E Value &1 for entry length parameter is not valid.
CPF3C0B E Value &1 for immediate update parameter is not valid.
CPF3C0C E Value &1 for key length parameter is not valid.
CPF3C0D E Value &1 for key insertion parameter is not valid.
CPF3C0E E Value &1 for the optimization parameter is not valid.
CPF3C03 E User index &2 not created.
CPD3C01 D Object name &1 is not valid.
CPD3C02 D Value &1 for entry length attribute parameter is not valid.
CPD3C03 D Extend attribute &1 is not valid.
CPD3C05 D Value &1 for authority parameter is not valid.
CPD3C0A D Value &1 for entry length parameter is not valid.
CPD3C0B D Value &1 for immediate update parameter is not valid.
CPD3C0C D Value &1 for key length parameter is not valid.
CPD3C0D D Value &1 for key insertion parameter is not valid.
CPD3C0E D Value &1 for the optimization parameter is not valid.
CPF3C2A E Value &1 for entry length attribute parameter is not valid.
CPF3C2B E Extended attribute &1 is not valid.
CPF3C2D E Value &1 for authority parameter is not valid.
CPF3C29 E Object name &1 is not valid.
CPF3C34 E Value &1 for replace option is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF3C45 E Value &1 not valid for domain parameter.
CPF3C49 E Request for user domain object cannot be granted.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.
CPF9870 E Object &2 type *&5 already exists in library &3.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Delete User Index (QUSDLTUI) API

Parameters

Required Parameter Group:

1	Qualified user index name	Input	Char(20)
2	Error code	I/O	Char(*)

The Delete User Index (QUSDLTUI) API deletes user indexes created with the Create User Index (QUSCRTUI) API. The QUSDLTUI API performs the same function as the Delete User Index (DLTUSRIDX) command.

Authorities and Locks

Library Authority *READ
User Index Authority *OBJEXIST and *USE
User Index Lock *EXCL

Required Parameter Group

Qualified user index name
INPUT; CHAR(20)
The name of the user index and the name of the library in which it resides. The first 10 characters contain the user index name, and the second 10 characters contain the library name. The user index name can be either a specific name or a generic name, a string of one or more characters followed by an asterisk (*). If you specify a generic name, QUSDLTUI deletes all user indexes that have names beginning with the string for which the user has authority.

You can use these special values for the library name:

***ALL** All libraries
***ALLUSR** All user-defined libraries, plus libraries containing user data and having names starting with Q
***CURLIB** The job's current library
***LIBL** The library list
***USRLIBL** The user portion of the job's library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF2105 E Object &1 in &2 type *&3 not found.
CPF2110 E Library &1 not found.
CPF2113 E Cannot allocate library &1.
CPF2114 E Cannot allocate object &1 in &2 type *&3.
CPF2117 E &4 objects type *&3 deleted. &5 objects not deleted.
CPF2125 E No objects deleted.
CPF2176 E Library &1 damaged.
CPF2182 E Not authorized to library &1.
CPF2189 E Not authorized to object &1 in &2 type *&3.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Remove User Index Entries (QUSRMVUI) API

Parameters

Required Parameter Group:

1	Number of entries removed	Output	Binary(4)
2	Entries removed	Output	Char(*)
3	Length of entries removed	Input	Binary(4)
4	Entry lengths and offsets	Output	Array(*) of Char(8)
5	Length of entry lengths and offsets	Input	Binary(4)
6	Returned library name	Output	Char(10)
7	Qualified user index name	Input	Char(20)
8	Format	Input	Char(8)
9	Maximum number of entries	Input	Binary(4)
10	Remove type	Input	Binary(4)
11	Remove criteria	Input	Char(*)
12	Length of remove criteria	Input	Binary(4)
13	Remove criteria offset	Input	Binary(4)
14	Error code	I/O	Char(*)

The Remove User Index Entries (QUSRMVUI) API removes one or more user index entries that match the values specified on the remove criteria parameter. It returns the number of entries that were removed and, optionally, returns the actual index entries removed.

Authorities and Locks

User Index Library Authority
*USE
User Index Authority
*CHANGE
User Index Lock *EXCL

Required Parameter Group

Number of entries removed
OUTPUT; BINARY(4)
The number of index entries, satisfying the values specified on the remove criteria parameter, that were successfully removed from the user index. If this field is 0, no entries satisfied the remove criteria. This value can never be greater than the maximum number of entries parameter.

Entries removed
OUTPUT; CHAR(*)
The actual entries removed. All entries that satisfied the remove criteria parameter and were removed (up to the maximum number of entries parameter) are returned if sufficient space is provided. The API returns only the data that the area can hold.

The size of the entries removed parameter should be greater than or equal to:

```
8 + (the maximum number of entries parameter
     * the maximum entry length)
```

The maximum entry length was defined when the index was created. It can be obtained by using the Retrieve User Index Attributes (QUSRUIAT) API.

To determine if all the entries are valid in the entries removed parameter, compare the bytes returned and the bytes available fields in the entries removed parameter.

The entries are always returned starting with the entry that is closest to or equal to the remove argument. Then entries are kept in the order that they proceed away from the remove criteria parameter. Each entry removed from the user index is based on the binary value of the remove criteria. No other collating sequence is supported. User indexes can contain only scalar data, which makes the index entries contiguous. Use the entry lengths and entry offsets parameter to parse the entries that were removed and returned in this parameter.

If you do not want the entries that were removed in this parameter to be returned, specify 0 for the length of entries removed parameter.

Every entry removed causes the number of entries removed parameter to be incremented by 1. You can also use the Retrieve User Index Attributes (QUSRUIAT) API to retrieve this information.

Refer to "IDXE0100 Format" on page 44-9 for the layout of this parameter.

Length of entries removed
INPUT; BINARY(4)
The length of the entries removed. If this length is larger than the size of the entries removed parameter, the results may not be predictable. The minimum length is 0 or ≥ 8 bytes. If 0 is used, the entries removed from the index are not returned and the bytes returned and the bytes available in the entries removed parameter are not set.

Entry lengths and entry offsets
OUTPUT; ARRAY of CHAR(8)
A data structure that contains entry lengths and entry offsets for all entries that were found that met the

remove criteria parameter. An entry length and entry offset exist for every entry returned in the entries removed parameter. These entry lengths and entry offsets are used to parse through the entries removed parameter. If the length of entries removed parameter is 0, this information will not be returned.

The size of the entry lengths and entry offsets parameter should be at least:

```
8 + (the maximum number of entries parameter * 8)
```

You must provide enough space in both the entries removed and the entry lengths and offset parameter for this API to return this information to you.

You will not receive complete information in the following two situations.

- You provide enough space in the entries removed parameter for the API to return all index entries removed, but there is not enough space in the entry lengths and entry offset parameter to return the lengths and offsets for all entries. You will be unable to parse through all of the entries in the entries removed parameter.
- You provide enough space in the entry lengths and entry offsets parameter to return all lengths and offsets, but there is not enough space in the entries removed parameter to return all index entries removed. Some of the entry lengths and entry offsets will not be valid; they will refer to index entries that could not be returned to you. Check the bytes returned and bytes available fields in the entries removed parameter to ensure that the information is complete.

See "Format for Entry Lengths and Entry Offsets" on page 44-9 for details on the data structure.

Length of entry lengths and entry offsets
INPUT; BINARY(4)
The length of the entry lengths and entry offsets. If the length is longer than the entry lengths and entry offsets parameter, the results may not be predictable. The minimum length is 8. If the length of entries removed parameter is 0, which means you do not want the entries removed to be returned, this parameter is ignored.

Returned library name
OUTPUT; CHAR(10)
The name of the library that contains the user index from which the entries were removed. If the entries are successfully removed from the user index, the name of the library that contained the user index entries is returned. This parameter is not set if an error occurs.

Qualified user index name
INPUT; CHAR(20)
The user index from which you want to remove entries, and the library in which it is located. The first 10 characters contain the user index 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
INPUT; CHAR(8)
The format of the user index entries that were removed. The format name supported is:

IDXE0100 Basic Information

Refer to "IDXE0100 Format" on page 44-9 for details on the format.

Maximum number of entries
INPUT; BINARY(4)
The maximum number of user index entries to be removed that satisfy the remove criteria. Valid values are 1 through 4095.

Remove type
INPUT; BINARY(4)
The type of remove operation that is to be performed. Valid values are:

1 Equal
Remove entries that are equal to the remove criteria.
2 Greater than
Remove entries that are greater than the remove criteria.
3 Less than
Remove entries that are less than the remove criteria.
4 Greater than or equal
Remove entries that are greater than or equal to the remove criteria.
5 Less than or equal
Remove entries that are less than or equal to the remove criteria.
6 First
Remove the first index entry or entries.
7 Last
Remove the last index entry or entries.
8 Between
Remove all entries between the two arguments specified in the remove criteria.

Remove criteria
INPUT; CHAR(*)
The criteria used to find matches in the user index.

When the remove type is 8 (between), this parameter contains two criteria elements of the same length. The first element is considered the starting element, and the second element is the ending element. This parameter is ignored when the remove type is 6 (first) or 7 (last).

Length of remove criteria
INPUT; BINARY(4)
The length of the remove criteria being used. This parameter is ignored when the remove type is 6 (first) or 7 (last). If the remove type is 8 (between), this parameter specifies the length of the first element. The

second element must have the same length as the first element. Valid values are 1–2000, depending on how the user index was created.

For a fixed and keyed user index, the length of the remove criteria can be greater than the length of the key.

Remove criteria offset
INPUT; BINARY(4)
The offset of the second element from the beginning of the remove criteria parameter. This parameter is ignored unless the remove type is 8 (between).

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format for Entry Lengths and Entry Offsets

The following information is returned in the entry lengths and entry offsets parameter. This information is needed to parse through the entries removed parameter. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
Note: The following fields will be repeated. The number of times they are repeated depends on the length of the entry lengths and entry offsets parameter and the number of entries actually removed.			
*	*	BINARY(4)	Entry length
*	*	BINARY(4)	Entry offset

IDXE0100 Format

The following information is returned for the IDXE0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(*)	Entry 1–n

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 the data actually returned.

Entry length. The length of the entry removed from the user index. Valid values are 1–2000, depending on how the user index was created.

Entry offset. The number of bytes from the beginning of the immediately preceding entry to the first byte of the entry returned. For the first entry, the offset is the number of bytes from the beginning of the parameter to the first byte of the first entry.

Entry 1–n. All entries that satisfy the remove criteria (up through the maximum number of entries) are returned. User indexes contain only scalar data, which makes the index entries contiguous. Use the entry length and entry offset values to parse this parameter.

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.
CPF3C7D E Remove or search information is not valid.
CPF3C70 E Length of entries removed parameter is not valid.
CPF3C76 E Length of lengths and offsets of entries &1 is not valid.
CPF3C77 E Remove type &1 is not valid.
CPF3C78 E Criteria length &1 is not valid.
CPF3C79 E Maximum number of entries &1 is not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve User Index Attributes (QUSRUIAT) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified user index name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve User Index Attributes (QUSRUIAT) API retrieves information about the current attributes and the current operational statistics of the user index.

Authorities and Locks

User Index Library Authority
*USE

User Index Authority
*USE

User Index Lock *SHRNUP

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. You can specify the size of this area to be smaller than the format requested if you specify the length of receiver variable parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the index information returned. The format name supported is:

IDXA0100 Basic information

Refer to "IDXA0100 Format" for details on the format.

Qualified user index name
INPUT; CHAR(20)
The user index for which you want to retrieve information, and the library in which it is located. The first 10 characters contain the user index name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

IDXA0100 Format

The following information is returned for the IDXA0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	User index name
18	12	CHAR(10)	User index library name
28	1C	CHAR(1)	Entry length attribute
29	1D	CHAR(1)	Immediate update
30	1E	CHAR(1)	Key insertion
31	1F	CHAR(1)	Optimized processing mode
32	20	CHAR(4)	Reserved
36	24	BINARY(4)	Entry length
40	28	BINARY(4)	Maximum entry length
44	2C	BINARY(4)	Key length
48	30	BINARY(4)	Number of entries added
52	34	BINARY(4)	Number of entries removed
56	38	BINARY(4)	Number of retrieve operations

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 the data actually returned.

Entry length. For user indexes with fixed-length entries, this is the length of each index entry. For user indexes with variable-length entries, this is equal to the longest entry that has ever been inserted into the index. Valid values are from 1 through 2000.

Entry length attribute. The types of entries in the user index. Possible values are:

F Fixed-length entries
V Variable-length entries

Immediate update. Whether or not the updates to the index are written synchronously to auxiliary storage on each update to the index. The possible values are:

0 No immediate update
1 Immediate update

Key insertion. Whether or not the inserts to the index are by key.

0 No insertion by key
1 Insertion by key

Key length. The length of the key where the first byte of an entry is the beginning of the key for the index entries. This field will be 0 for a nonkeyed user index.

Maximum entry length. The maximum entry length any user index entry can have.

Number of entries added. The number of entries added to the user index. The number of entries currently in the index can be obtained by subtracting the number of entries removed from the number of entries added.

Number of entries removed. The number of entries removed from the user index.

Number of retrieve operations. The number of times either the FNDINXEN (find independent index entry) MI instruction or Retrieve User Index Entry (QUSRTVUI) API has been used on this user index. The QUSRUIAT API or MATINXAT (materialize independent index attributes) MI instruction sets the number of retrieve operations to 0 after the retrieve or materialize operation is completed.

Optimized processing mode. Whether the user index is maintained in a manner that optimizes performance for:

0 Random references
1 Sequential references

Reserved. An ignored field.

User index library name. The name of the library containing the user index. This information is helpful when *CURLIB or *LIBL is specified in the qualified user index name parameter.

User index name. The name of the user index.

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.
CPF3C24 E Length of the receiver variable is not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve User Index Entries (QUSRTVUI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Entry lengths and offsets	Output	Array(*) of Char(8)
4	Length of entry lengths and offsets	Input	Binary(4)
5	Number of entries returned	Output	Binary(4)
6	Returned library name	Output	Char(10)
7	Qualified user index name	Input	Char(20)
8	Format	Input	Char(8)
9	Maximum number of entries	Input	Binary(4)
10	Search type	Input	Binary(4)
11	Search criteria	Input	Char(*)
12	Length of search criteria	Input	Binary(4)
13	Search criteria offset	Input	Binary(4)
14	Error code	I/O	Char(*)

The Retrieve User Index Entries (QUSRTVUI) API retrieves user index entries that match the criteria specified on the search criteria parameter.

The entries are always returned starting with the entry that is closest to or equal to the search criteria parameter and then proceeding away from the search criteria. The number of entries returned parameter will never exceed the value specified in the maximum number of entries parameter. Each entry retrieved from the user index is based on the binary value of the search criteria parameter. No other collating sequence is supported.

Every entry retrieved causes the number of retrieve operations to be incremented by 1.

Authorities and Locks

User Index Library Authority *USE
User Index Authority *USE
User Index Lock *SHRNUP

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. You can specify the size of this area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold.

Use the entry lengths and entry offsets parameter to parse through this parameter. If the number of entries returned parameter is 0, then only the bytes available and the bytes provided have been changed.

To determine if all the entries are valid in the receiver variable, compare the bytes returned and bytes available fields. If the bytes returned are less than the bytes available, your receiver variable is not large enough to hold all the entries that match the search criteria parameter. While processing the entries, you need to make sure that both:

- Your current offset in the receiver variable plus the entry offset is less than the length of the receiver variable
- Your current offset in the receiver variable plus the entry offset plus the entry length is less than the length of the receiver variable.

The size of the receiver variable parameter should be greater than or equal to:

```
8 + (the maximum number of entries parameter
     * the maximum entry length)
```

The maximum entry length was defined when the index was created. It can be obtained by using the Retrieve User Index Attributes (QUSRUIAT) API.

Refer to the "IDXE0100 Format" on page 44-13 for the layout of this parameter.

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.

Entry lengths and entry offsets
OUTPUT; ARRAY(*) of CHAR(8)
A data structure containing entry lengths and entry offsets for all entries found that met the search criteria. An entry length and entry offset exist for every entry returned in the receiver variable. These entry lengths and entry offsets are used to parse through the receiver variable.

The size of the entry lengths and entry offsets parameter should be at least:

```
8 + (the maximum number of entries parameter * 8)
```

You must provide enough space in both the receiver variable and the entry lengths and entry offsets parameter for this API to return this information to you. You will not receive complete information in the following two situations.

- You provide enough space in the receiver variable parameter for the API to return all index entries, but there is not enough space in the entry lengths and entry offset parameter to return the lengths and offsets for all entries. You will be unable to parse through all of the entries in the receiver variable parameter.
- You provide enough space in the entry lengths and entry offsets parameter to return all lengths and offsets, but there is not enough space in the receiver variable parameter to return all index entries removed. Some of the entry lengths and entry offsets will not be valid; they will refer to index entries that could not be returned to you. Check the bytes returned and bytes available fields in the receiver variable parameter to ensure that the information is complete.

See the "Format for Entry Lengths and Entry Offsets" on page 44-13 for details about the data structure.

Length of entry lengths and entry offsets
INPUT; BINARY(4)
The length of the entry lengths and entry offsets parameter. If the length is longer than the entry lengths and entry offsets parameter, the results may not be predictable. The minimum length is 8.

If the receiver variable cannot hold all the entries that satisfy the search criteria:

- The entries are truncated.
- Not all the information in the entry lengths and entry offsets parameter is valid.

Number of entries returned
OUTPUT; BINARY(4)
The total number of index entries found that satisfy the search criteria. If this field is 0, no entries satisfied the search criteria. This value can never be greater than the maximum number of entries parameter.

Returned library name
OUTPUT; CHAR(10)
The name of the library that contains the user index from which the entries were successfully retrieved. This parameter is not set if an error occurs.

Qualified user index name
INPUT; CHAR(20)
The user index for which you want to retrieve information, and the library in which it is located. The first 10 characters contain the user index 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
INPUT; CHAR(8)
The format of the receiver variable. The format name supported is:

IDXE0100 Basic Information

Refer to "IDXE0100 Format" on page 44-13 for details about the format.

Maximum number of entries
INPUT; BINARY(4)
The maximum number of index entries to be returned

that match the search criteria. Valid values are 1 through 4095.

Search type
INPUT; BINARY(4)
The type of search that is to be performed. Valid values are:

1 Equal
Find entries equal to the search criteria.
2 Greater than
Find entries that are greater than the search criteria.
3 Less than
Find entries that are less than the search criteria.
4 Greater than or equal
Find entries that are greater than or equal to the search criteria.
5 Less than or equal
Find entries that are less than or equal to the search criteria.
6 First
Find the first index entry or entries.
7 Last
Find the last index entry or entries.
8 Between
Find all entries between the two arguments specified in the search criteria.

Search criteria
INPUT; CHAR(*)
The criteria used to find matches in the user index.

If the search type is 8 (between), both search elements must have the same length. When the search type is 8 (between), this parameter contains two search elements. The first element is considered the starting element, and the second element is the ending element.

This parameter is ignored when the search type parameter is 6 (first) or 7 (last).

Length of search criteria
INPUT; BINARY(4)
The length of the search criteria that is to be used. This parameter is ignored when the search type is 6 (first) or 7 (last).

If the search type is 8 (between), this parameter specifies the length of the first element. The second element must have the same length as the first element. Valid values are 1–2000, depending on how the user index was created.

For a fixed and keyed user index, the length of the search criteria:

- Can be greater than the length of the key
- Must be less than or equal to the entry length

Search criteria offset
INPUT; BINARY(4)
The offset of the second search element from the beginning of the search criteria parameter. This parameter is ignored unless the search type is 8 (between).

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format for Entry Lengths and Entry Offsets

The following information is returned in the entry lengths and entry offsets parameter. The information is needed to parse through the receiver variable. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
Note: The following fields will be repeated. The number of times they are repeated depends on the length of the entry lengths and entry offsets parameter and the number of entries actually retrieved.			
*	*	BINARY(4)	Entry length
*	*	BINARY(4)	Entry offset

IDXE0100 Format

The following index information is returned for the IDXE0100 format in the receiver variable parameter. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(*)	Entry 1–n

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 the data actually returned.

Entry length. The length of the entry retrieved from the index. Valid values are 1–2000, depending on how the user index was created.

Entry offset. The number of bytes from the beginning of the immediately preceding entry to the first byte of the entry returned. For the first entry, the offset is the number of bytes

from the beginning of the receiver variable to the first byte of the first entry.

Entry 1–n. All entries that satisfy the search criteria (up to the maximum number of entries parameter) are returned. User indexes are created to contain only scalar data, which results in the index entries being contiguous. Use the entry length and entry offset values to parse this field.

This field is repeated by the value in the number of entries returned parameter if the receiver variable is large enough to hold all of the entries found.

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.
CPF3C24 E Length of the receiver variable is not valid.
CPF3C7A E Search type &1 is not valid.
CPF3C7D E Remove or search information is not valid.
CPF3C76 E Length of lengths and offsets of entries &1 is not valid.
CPF3C78 E Criteria length &1 is not valid.
CPF3C79 E Maximum number of entries &1 is not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Chapter 45. User Queue APIs

This chapter describes the user queue APIs, which let you create and delete user queues. **User queues** are permanent objects with an object type of *USRQ. They provide a way for one or more processes to communicate asynchronously.

You can use user queues to:

- Communicate between two processes asynchronously
- Store data in arrival sequence for later use
- Contain keyed messages
- Use the batch compile machine
- Control busy resources
- Permit better performance than the data queue interface
- Process a queue by key rather than by FIFO (first-in first-out) or LIFO (last-in first-out)

You can save and restore user queues. However, you can only save or restore its definition. You cannot save or restore the messages in it. You cannot restore a user queue if a user queue with the same name already exists in the library. You must provide programs to use this object type to enqueue and dequeue messages.

The user queue APIs are:

Create User Queue (QUSCRTUQ)
Delete User Queue (QUSDLTUQ)

In addition to using these APIs, you can work with user queues through the System C/400 PRPQ, machine interface (MI) instructions, and the Delete User Queue (DLTUSRQ) command. For details about MI instructions, see the *Languages: System C/400 Programming RPQ P10102 User's Guide and Reference* or the *MI Functional Reference*. For details about the DLTUSRQ command, see the *CL Reference*.

Create User Queue (QUSCRTUQ) API

Parameters

Required Parameter Group:

1	Qualified user queue name	Input	Char(20)
2	Extended attribute	Input	Char(10)
3	Queue type	Input	Char(1)
4	Key length	Input	Binary(4)
5	Maximum message size	Input	Binary(4)
6	Initial number of messages	Input	Binary(4)
7	Additional number of messages	Input	Binary(4)
8	Public authority	Input	Char(10)
9	Text description	Input	Char(50)

Optional Parameter Group 1:

10	Replace	Input	Char(10)
11	Error code	I/O	Char(*)

Optional Parameter Group 2:

12	Domain	Input	Char(10)
13	Pointers	Input	Char(10)

The Create User Queue (QUSCRTUQ) API creates a user queue in either the user domain or the system domain, based on the input parameters. A user-domain user queue can be directly manipulated with MI instructions. If you are running at security level 40 or greater, you cannot directly manipulate a system-domain user queue using MI instructions.

When user queues are created, they:

- Can grow to a maximum of 16 megabytes in size, though some of that is used for the queue definition.
- Are placed in the same auxiliary storage pool (ASP) as the library.

The system can automatically extend user queues, so you should create a small queue and allow the system to extend the queue as needed. Smaller queues provide better performance and use less storage. However, the system does not automatically reduce the size of the queue if it is too large. If you specify the additional number of messages as 0, the system does not extend the user queue you create when the queue is full.

To decrease the size of a user queue, a user must delete the queue and create it again.

There are no APIs to manipulate user queue entries. If you must access a user queue object in a library that does not permit user-domain user objects, you must use data queue objects. For information about data queues, refer to the *CL Programmer's Guide*.

If you need to know into which domain the user queue object was created, use either of the following APIs to retrieve that information:

- Retrieve Object Description (QUSROBJD) API
- List Object (QUSLOBJ) API

Authorities and Locks

Library Authority
*OBJOPR, *ADD, and *READ.

User Queue Authority
*OBJMGT, *OBJEXIST, and *READ. These authorities are required only if the replace parameter is used and if there is an existing user queue to replace.

User Queue Lock
*EXCL. This applies to both the user queue being created and an existing user queue being replaced.

Required Parameter Group

Qualified user queue name
INPUT; CHAR(20)
The first 10 characters contain the user queue name, and the second 10 characters contain the name of the library where the user queue is located. The only special value supported is *CURLIB.

User queues created in the QTEMP and QRPLOBJ libraries are not forced to permanent storage; they are deleted when those libraries are cleared at sign-off and system IPL, respectively.

Extended attribute
INPUT; CHAR(10)
The extended attribute of the user queue. For example, an object type of *FILE has an extended attribute of PF (physical file), LF (logical file), DSPF (display file), SAVF (save file), and so on.

The extended attribute must be a valid *NAME. You can enter this parameter in uppercase, lowercase, or mixed case. The API automatically converts it to uppercase.

Queue type
INPUT; CHAR(1)
The sequence in which messages are to be dequeued from the queue. The valid values are:

F First-in first-out
K Keyed
L Last-in first-out

Key length
INPUT; BINARY(4)
The length in bytes of the message key from 1 to 256 if you specify the queue type as keyed. If you specify that the queue is not a keyed queue, the value must be 0.

Maximum message size
INPUT; BINARY(4)
The maximum allowed size of messages to be placed on the message queue. The API truncates messages that are longer than the value you specify. The maximum size allowed is 64 000 bytes.

Initial number of messages
INPUT; BINARY(4)
The initial number of messages that the queue can contain.

Additional number of messages
INPUT; BINARY(4)
The amount to increase the maximum number of messages value when the queue is full. When this parameter is set to 0, the queue is not extended if an overflow occurs and an error message is sent to the program attempting to place a message on the queue.

Public authority
INPUT; CHAR(10)
The authority you give to the users who do not have specific private or group authority to the user queue. Once the user queue has been created, its public authority stays the same when it is moved to another library or restored from backup media.

If the replace parameter is used and a user queue exists to be replaced, this parameter is ignored. All authorities are transferred from the replaced user queue to the new one.

The valid values for this parameter are:

***ALL** The user can perform all authorized operations on the user queue.

Authorization list name
The user queue is secured by the specified authorization list, and its public authority is set to *AUTL. The specified authorization list must exist on the system when this API is issued. If it does not exist, the create process fails, and an error message is returned to the application.

***CHANGE**
The user has read, add, update, and delete authority to the user queue and can read the object description.

***EXCLUDE**
The user cannot access the user queue in any way.

***LIBCRTAUT**
The public authority for the user queue is taken from the CRTAUT value for the target library when the object is created. If the CRTAUT value for the library changes later, that change does not affect user queues already created. If the CRTAUT value contains an authorization list name and that authorization list secures an object, do not delete the list. If you do, the next time you call this API with the *LIBCRTAUT parameter, it will fail.

***USE** The user can read the object description and the user queue's contents but cannot change them.

Text description
INPUT; CHAR(50)
A brief description of the user queue.

Optional Parameter Group 1

Replace
INPUT; CHAR(10)
Whether to replace an existing user queue. Valid values for this parameter are:

***NO** Do not replace an existing user queue of the same name and library. *NO is the default.
***YES** Replace an existing user queue of the same name and library.

The user queue being replaced is destroyed if both:

- The user queue you are replacing is in the user domain.
- The allow user domain (QALWUSRDMN) system value is not set to *ALL or does not contain the library QRPLOBJ.

If the QRPLOBJ library is specified in the QALWUSRDMN system value, then the replaced user-domain user queue is moved to the QRPLOBJ library. If the user queue is in the system domain, it is moved to the QRPLOBJ library. System domain objects can exist in any library that is cleared at system IPL. For details about authorities, ownership, and renaming, see the discussion of the REPLACE parameter in Volume 3 of 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. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Optional Parameter Group 2

Domain
INPUT; CHAR(10)
The domain into which the user queue should be created. If this parameter is not specified, the value of *DEFAULT will be assumed by the API. Valid values for this parameter are:

***DEFAULT**
Allows the system to decide into which domain the object should be created.
***SYSTEM**
Creates the user queue object into the system domain. The API can always create a user queue into the system domain regardless of the security level you are running at. However, system-domain user queues can only be used at security level 30 and below because there are no APIs to access user queues.
***USER**
Attempts to create the user queue object into the user domain. This is not always possible. If the library you are creating the user queue into does not appear in the QALWUSRDMN system value, the API cannot create the user queue into the user domain. An error message will be returned.

The API uses the following criteria to determine into which domain to create the user queue. The destination library is the library you specified in the qualified user queue name parameter. The optional domain parameter is the information specified in the domain parameter.

QALWUSRDMN System Value	Destination Library	Optional Domain Parameter	Domain of Created Object
*ALL	Any	*DEFAULT	User domain (no change)
*ALL	Any	*SYSTEM	System domain
*ALL	Any	*USER	User domain
QTEMP	QTEMP	*DEFAULT	User domain
QTEMP	QTEMP	*SYSTEM	System domain
QTEMP	QTEMP	*USER	User domain
Does not contain library name	Library name	*DEFAULT	System domain
Does not contain library name	Library name	*SYSTEM	System domain
Does not contain library name	Library name	*USER	None; error is returned

Note: The QALWUSRDMN system value lists the libraries into which the user domain objects can be created. The libraries include special value *ALL or a list of one or more library names.

Pointers
INPUT; CHAR(10)
Whether or not the user queue messages can contain pointer data or not. If this parameter is not specified, *NO is assumed.

***YES** Messages can contain pointer and scalar data. Messages to be enqueued must be 16-byte aligned, regardless of whether or not the message contains pointer data. Only user-domain user queues can contain pointer data; therefore, queues that support pointers cannot be created in or restored to a library that is not permitted by system value QALWUSRDMN. User queues that can

contain pointers cannot be saved for a release prior to Version 2 Release 3 Modification Level 0.

***NO** Messages can contain scalar data only. (User queues created prior to Version 2 Release 3 Modification Level 0 contain scalar data only). The user queue can be in either the system domain or the user domain.

Error Messages

CPF2143 E Cannot allocate object &1 in &2 type *&3.
CPF2144 E Not authorized to &1 in &2 type *&3.
CPF2283 E Authorization list &1 does not exist.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C02 E User queue &1 not created.
CPD3C01 D Object name &1 is not valid.
CPD3C03 D Extended attribute &1 is not valid.
CPD3C05 D Value &1 for authority parameter is not valid.
CPD3C06 D Number of messages requested, &1, is too large for this queue.
CPD3C07 D Value &1 for queue type parameter not valid.
CPD3C08 D Initial number of queue messages specified, &1, is not valid.
CPD3C09 D Extension parameter value &1 for queue overflow is not valid.
CPD3C10 D Value &1 for key length parameter is not valid.
CPD3C11 D Value &1 for maximum message size parameter is not valid.
CPF3C08 E Initial number of queue messages specified, &1, is not valid.
CPF3C09 E Extension parameter value &1 for queue overflow is not valid.
CPF3C10 E Value &1 for key length parameter is not valid.
CPF3C11 E Value &1 for maximum message size parameter is not valid.
CPF3C2B E Extended attribute &1 is not valid.
CPF3C2D E Value &1 for authority parameter is not valid.
CPF3C2E E Number of messages requested, &1, is too large for this queue.
CPF3C2F E Value &1 for queue type parameter not valid.
CPF3C29 E Object name &1 is not valid.
CPF3C34 E Value &1 for replace option is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF3C45 E Value &1 not valid for domain parameter.
CPF3C46 E Value &1 not valid for pointers parameter.
CPF3C47 E Pointers not valid for system domain user queue.
CPF3C49 E Request for user domain object cannot be granted.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.
CPF9870 E Object &2 type *&5 already exists in library &3.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Delete User Queue (QUSDLTUQ) API

Parameters

Required Parameter Group:

1	Qualified user queue name	Input	Char(20)
2	Error code	I/O	Char(*)

The Delete User Queue (QUSDLTUQ) API deletes user queues created with the Create User Queue (QUSCRTUQ) API. The Delete User Queue (DLTUSRQ) command has the same function.

Authorities and Locks

Library Authority *READ
User Queue Authority *OBJEXIST and *USE
User Queue Lock *EXCL

Required Parameter Group

Qualified user queue name
INPUT; CHAR(20)
The name of the user queue and the name of the library in which it resides. The first 10 characters contain the user queue name, and the second 10 characters contain the library name.

The user queue name can be either a specific name or a generic name, a string of one or more characters followed by an asterisk (*). If you specify a generic name, QUSDLTUS deletes all user queues that have names beginning with the string for which the user has authority.

You can use these special values for the library name:

***ALL** All libraries
***ALLUSR** All user-defined libraries, plus libraries containing user data and having names starting with Q
***CURLIB** The job's current library
***LIBL** The library list
***USRLIBL** The user portion of the job's library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF2105 E Object &1 in &2 type *&3 not found.
CPF2110 E Library &1 not found.
CPF2113 E Cannot allocate library &1.
CPF2114 E Cannot allocate object &1 in &2 type *&3.
CPF2117 E &4 objects type *&3 deleted. &5 objects not deleted.
CPF2125 E No objects deleted.
CPF2176 E Library &1 damaged.
CPF2182 E Not authorized to library &1.
CPF2189 E Not authorized to object &1 in &2 type *&3.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Chapter 46. Object APIs

You can use object APIs to obtain information about OS/400 objects. The object APIs and their functions include the following:

Change Library List (QLICHGLL) changes the current library, the two product libraries, and the user part of the job's library list.
Change Object Description (QLICOBJD) changes object information for a specific object.
Convert Type (QLICVTTP) converts an object type to and from hexadecimal format.
List Objects (QUSLOBJ) generates a list of object names and descriptive information based on the specified parameters.
Rename Object (QLIRNMO) renames an existing object to a new object name or new library name or both and optionally replaces the object.
Retrieve Object Description (QUSROBJD) retrieves object information for a specific object.

The QUSLOBJ and QUSROBJD APIs return much of the same information. However, the APIs differ in several respects:

- The APIs group the returned information differently among their output formats. For example, the OBJL0300 format of the QUSLOBJ API does not contain exactly the same data as the OBJD0300 format of the QUSROBJD API.
- The APIs use different data formats for some specific items, such as dates and times.
- The APIs differ in efficiency, depending on your application. In most cases, the QUSROBJD API is faster at retrieving information about a single object. The QUSLOBJ API is faster at retrieving information about several objects.

The APIs in this chapter are presented in alphabetical order.

Change Library List (QLICHGLL) API

Parameters

Required Parameter Group:

1	Current library name	Input	Char(11)
2	First product library name	Input	Char(11)
3	Second product library name	Input	Char(11)
4	User library list names	Input	Array(*) of Char(11)
5	Number of user library names	Input	Binary(4)
6	Error code	I/O	Char(*)

The Change Library List (QLICHGLL) API changes the current library, the two product libraries and the user part of the job's library list.

When the library list is changed, each library added to the list is locked with a shared-read lock. If a library is being removed from the library list, the shared-read lock is released.

You may want to save the job's library list before changing it. The Retrieve Job Information (QUSRJOBI) API can be used to do this. You can then change the library list back to its original value using the QLICHGLL API with the libraries saved from the QUSRJOBI API.

Authorities and Locks

Library Authority *USE
Library Lock *SHRRD

Required Parameter Group

Current library name
INPUT; CHAR(11)
The library that replaces the current library in the job's library list. (The data is left-justified with a blank at the end.)

The current library can be, but does not have to be, a duplicate of any library in the library list. QTEMP cannot be specified for the library name.

The following special values can be used:

***CRTDFT** No library should be in the current library entry in the library list. If objects are created into the current library, then library QGPL is used as the current default library.
***SAME** The current library in the job's library list is not changed.

First product library name
INPUT; CHAR(11)
The library that replaces the first product library entry in the job's library list. (The data is left-justified with a blank at the end.)

A product library may be a duplicate of the current library or of a library in the user part of the library list.

The following special values can be used:

***NONE** The first product library entry in the job's library list is removed.
***SAME** The first product library entry in the job's library list is not changed.

Second product library name
INPUT; CHAR(11)
The library that replaces the second product library entry

in the job's library list. (The data is left-justified with a blank at the end.)

The following special values can be used:

***NONE** The second product library entry in the job's library list is removed.
***SAME** The second product library entry in the job's library list is not changed.

User library list names
INPUT; ARRAY(*) of CHAR(11)
The libraries that replace the libraries in the user part of the job's library list. Specify the names of the libraries in the order in which they are to be searched. (The data is left-justified with a blank at the end.)

The same library name cannot be specified more than once. A library cannot exist in the system part and the user part of the job's library list.

Number of user library names
INPUT; BINARY(4)
The total number of library names that will be changed in the user part of the job's library list.

This must be a value from -1 through 25. There is a limit of 25 libraries in the user part of the job's library list. When either of the following values is specified, the user library list names parameter is ignored:

-1 The user part of the library list remains the same.
0 All libraries in the user part of the library list are removed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF2106 E Library list not changed.
CPF2110 E Library &1 not found.
CPF2113 E Cannot allocate library &1.
CPF2133 E First product library on library list destroyed.
CPF2134 E Second product library on library list destroyed.
CPF2137 E Current library on library list destroyed.
CPF2176 E Library &1 damaged.
CPF2182 E Not authorized to library &1.
CPF2184 E Library list not replaced.
CPF219A E Library QTEMP cannot be specified.
CPF219F E Number of libraries for library list not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Change Object Description (QLICOBJD) API

Parameters

Required Parameter Group:

1	Returned library name	Output	Char(10)
2	Object and library name	Input	Char(20)
3	Object type	Input	Char(10)
4	Changed object information	Input	Char(*)
5	Error code	I/O	Char(*)

The Change Object Description (QLICOBJD) API lets you change object information for a specific object.

Before the object can be changed with the API, the allow change by program field is checked. If the API cannot be used to change the object, message CPF219B is issued.

When an object has been successfully updated by the API, the date and time the object was changed and the changed by program field are updated.

Authorities and Locks

Library Authority *READ
Non-*FILE Object Authority *OBJMGT
***FILE Object Authority** *OBJOPR and *OBJMGT
Object Lock *EXCLRD

Required Parameter Group

Returned library name
OUTPUT; CHAR(10)
The name of the library that contains the changed object. If *CURLIB, *LIBL, or a name is specified for the library name in the object and library name parameter, the value returned is the name of the library where the object was found. If an error occurred while the API was accessing the object, blanks are returned.

Object and library name
INPUT; CHAR(20)
The object for which you want to change information and the library in which it is located. The first 10 characters contain the object name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The job's library list

Object type
INPUT; CHAR(10)
The type of object for which you want to change the information. You can only specify specific external object types. An asterisk (*) must precede the object type. For a complete list of the available object types, see the *CL Reference*.

Changed object information
INPUT; CHAR(*)
The information for the object that you want to change. The information must be in the following format:

Number of variable length records
BINARY(4)
Total number of all of the variable length records.

Variable length records
The fields of the object's description to change and the data used for the change. For the specific format of the variable length record, see "Format for Variable Length Record."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format for Variable Length Record

The following table defines the format for the variable length records.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Key
4	4	BINARY(4)	Length of data
8	8	CHAR(*)	Data

If the length of the data is longer than the key field's data length, the data will be truncated at the right. No message will be issued.

If the length of the data is smaller than the key field's data length, the data will be padded with blanks at the right. No message will be issued.

Field Descriptions

Data. The data used to change a specific field of the object description. Validity checking is done on the values specified for the following keys to verify that they are 0 or 1.

- Allow change by program
- Days used count reset
- Last used date
- Changed date and time stamp

The data specified for other keys is not validity checked.

Key. Identifies a field of the object's description to change. Only specific fields of the object's description can be changed. See "Keys" for the list of valid keys.

Length of data. The length of the data used to change a specific field of the object's description.

Keys: The following table lists the valid keys for the key field area of the variable length record.

Key	Type	Field
1	CHAR(30)	Source file
2	CHAR(13)	Source file last changed date and time
3	CHAR(13)	Compiler
4	CHAR(8)	Object control level
5	CHAR(13)	Licensed program
6	CHAR(7)	Program temporary fix (PTF)
7	CHAR(6)	Authorized program analysis report (APAR)
8	CHAR(1)	Allow change by program
9	CHAR(10)	User-defined attribute
10	CHAR(50)	Text
11	CHAR(1)	Days used count reset
12	CHAR(4)	Product option load ID
13	CHAR(4)	Product option ID
14	CHAR(4)	Component ID
15	CHAR(1)	Last used date
16	CHAR(1)	Changed date and time stamp

Field Descriptions

Allow change by program. Whether to prevent users from changing an object's description with this API. It must have a value of 0 or 1.

0 Changes are not allowed with the API. Once this field has been changed to 0, the API cannot be used to make any further changes to the object's description.
1 Changes are allowed with the API. The API can be used to change the object's description.

Authorized program analysis report (APAR). The authorized program analysis report identification that caused this object to be patched. IBM APARs have an uppercase alphabetic character followed by 5 decimal numbers. If you want to conform with the system, this format should be followed.

Changed date and time stamp. The date and time the object was last changed. This is useful if you only want to update an object's change date/time stamp and do not want to update any other key fields.

0 The changed date and time stamp is not updated.
1 The changed date and time stamp is updated to the current system date and time.

This key cannot be specified with any other key. If it is, message CPF21A6 is issued.

Note: For database files, the last change date/time is updated for all members in the file and the file itself.

Compiler. The name, version level, release level, and modification level of the compiler. Objects created with IBM products will have the licensed program name of the compiler in the compiler name field and a version field in the VxRxMy

format where x must be 0-9 and y must be 0-9 or A-Z. If you want to conform with the system, this format should be followed.

Compiler name CHAR(7)
Version CHAR(6)

Component ID. The product administrator owns this field. It can be used to track information about objects, such as object size, at a lower level than the product option ID.

Days used count reset. This field is used to:

- Reset the number of days an object has been used on the system
- Update the date the days used count was last reset to 0.

It must have a value of 0 or 1.

0 Neither the days used count nor the date used count reset field is updated.
1 The days used count is set to 0. The date used count reset field is updated to the current system date.

Key fields 11 and 15 cannot both be specified. These changes are incompatible because key 11 will change the days used count to 0 and key 15 will increase the days used count. If both keys 11 and 15 are specified, message CPF21A1 is issued.

Note: For database files, the date the days used count was last reset and the days used count are updated for all members in the file.

Last used date. This field is used to:

- Update the last used date to the current system date.
- Increase the days used count.

It must have a value of 0 or 1.

0 The last used date field is not updated. The field is not increased.
1 The last used date field is updated to the current system date. If this is the first use of the object today (since midnight), the days used count reset field is increased.

Note: For database files, the last used date and number of days used count are updated for all members in the file.

The last used date field cannot be changed for a database file that does not have any members. Message CPF21A2 will be issued.

Licensed program. The name, version level, release level, and modification level of the licensed program. Objects that are a part of an IBM licensed program have a valid licensed program name (7 characters containing 0-9 and uppercase A-Z). The version field is in the VxRxMy format where x must be 0-9 and y must be 0-9 or A-Z. If you want to conform with the system, this format should be followed.

Licensed program name CHAR(7)
Version CHAR(6)

Object control level. The object control level for the object. IBM programs will have an 8-character decimal value.

Product option ID. Identifies part of a licensed program (product). Products can have multiple options. Objects that are a part of an IBM licensed program must be 0 (*BASE) through 99. If you want to conform with the system, this format should be followed.

Product option load ID. The language identifier associated with the object. Objects that are a part of an IBM licensed program must have one of the allowed languages in the 29xx format. If you want to conform with the system, this format should be followed.

Program temporary fix. The program temporary fix (PTF) that resulted in the creation of the object. For IBM objects the first 2 characters are a prefix ID, and the remaining 5 characters are the program change ID (decimal). The field is blank if the object was not changed because of a PTF. If you want to conform with the system, this format should be followed.

Source file. The source file used to create the object, the library name in which it is located, and the name of the source file member.

Source file name CHAR(10)
Library name CHAR(10)
Member name CHAR(10)

Objects created with IBM products have valid object names for the qualified source file name. If you want to conform with the system, this format should be followed.

Source file last changed date and time. The date and time the member in the source file was last updated. Objects created with IBM products will be in the CYYMMDDHHMMSS format, where:

C Century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
YY Year
MM Month
DD Day
HH Hour
MM Minute
SS Second

If you want to conform with the system, this format should be followed.

Text. The user-defined text that briefly describes the object and its function.

User-defined attribute. An attribute you define. This should not be confused with the extended attribute of the object. The extended attribute is set by the system when an object is created.

Error Messages

CPF21A1 E Last used date for &1 in &2 type *&3 cannot be changed.
CPF21A2 E Last used date for &1 in &2 type *FILE cannot be changed.
CPF21A6 E Cannot specify field &1 with any other fields.
CPF2150 E Object information function failed.
CPF2151 E Operation failed for &2 in &1 type *&3.
CPF219B E Cannot change &1 in &2 type *&3.
CPF219E E Object type *&1 not valid external object type.
CPF2199 E &2 not valid for field &1.
CPF24B4 E Severe error while addressing parameter list.
CPF2451 E Message queue &1 is allocated to another job.
CPF3CF1 E Error code parameter not valid.
CPF36F7 E Message queue QSYSOPR is allocated to another job.
CPF7304 E File &1 in &2 not changed.
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.
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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Convert Type (QLICVTTP) API

Parameters

Required Parameter Group:

1	Conversion	Input	Char(10)
2	Symbolic object type	I/O	Char(10)
3	Hexadecimal object type	I/O	Char(2)
4	Error code	I/O	Char(*)

The Convert Type (QLICVTTP) API lets you convert an object type from the external symbolic format to the internal hexadecimal format and vice versa.

You can use the QLICVTTP API to:

- Convert a specific symbolic object type to an equivalent hexadecimal object type
- Convert a specific hexadecimal object type to an equivalent symbolic object type

Required Parameter Group

Conversion
INPUT; CHAR(10)
The type of conversion to perform.

***HEXTOSYM** An object type in hexadecimal form is converted to an equivalent symbolic object type.
***SYMTOHEX** A symbolic object type is converted to an equivalent hexadecimal form.

Symbolic object type
I/O; CHAR(10)
The external symbolic name given to an object type. An asterisk (*) precedes the object type. The value of the conversion parameter specified determines if this is an input or output field. For a complete list of the available object types, see the *CL Reference*.

Hexadecimal object type
I/O; CHAR(2)
The MI representation of an external object type. This field is in hexadecimal form. The value of the conversion specified determines if this is an input or output field. For a list of the available object types in hexadecimal format, see the section headed "OS/400 Object Types and Subtypes" in the *Diagnostic Aids – Volume 1*.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF2101 E Object type *&1 not valid.
CPF2102 E Object type and subtype code &1 not valid.
CPF219C E Conversion value &1 not valid.
CPF219D E Object type &1 not valid external object type.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Objects (QUSLOBJ) API

Parameters

Required Parameter Group:

1	Qualified user space object	Input	Char(20)
2	Format name	Input	Char(8)
3	Object and library name	Input	Char(20)
4	Object type	Input	Char(10)

Optional Parameter:

5	Error code	I/O	Char(*)

The List Objects (QUSLOBJ) API lets you generate a list of object names and descriptive information based on specified selection parameters. The QUSLOBJ API places the list in

the specified user space. The generated list replaces any existing list in the user space.

You can use the QUSLOBJ API to:

- List objects in a library
- List objects of only one type
- Write an application program to move programs from the QRPLOBJ library back to where they were originally located
- Provide backup analysis based on when the object was last saved or last updated
- Provide source member and object analysis from source member information to verify that the current source was used to create the specified object

The QUSLOBJ API returns information in several formats. All formats except OBJL0100 include an information status field that describes the completeness and validity of the information. Be sure to check the information status field before using any other information returned.

Authorities and Locks

User Space Authority
*CHANGE

User Space Library Authority
*USE

User Space Lock *EXCLRD

Object Authority Any object authority except *EXCLUDE

Object Library Authority
*READ

Required Parameter Group

Qualified user space object
INPUT; CHAR(20)
The name of the *USRSPC object that is to receive the generated list. The first 10 characters contain the user space object name, and the second 10 characters contain the name of the library where the user space is located. The special values supported for the library name are *LIBL and *CURLIB.

Format name
INPUT; CHAR(8)
The format of the information returned on each object that is requested. You must use one of the following format names:

OBJL0100 Object names (fastest)
OBJL0200 Text description and extended attribute
OBJL0300 Basic object information
OBJL0400 Creation information
OBJL0500 Save and restore information
OBJL0600 Usage information
OBJL0700 All object information (slowest)

For details about the formats, see "Format of the Generated Lists." For performance reasons, you should choose the format that returns only as much information as you need. The higher the number of the format name, the more information is returned and the more time it takes to process.

Object and library name
INPUT; CHAR(20)
The object and library names to place in the *USRSPC object. The first 10 characters contain the object name, which may be a simple name, a generic name, or the special value *ALL.

The second 10 characters identify the name of the library or libraries to search for the specified objects. The following special values are allowed:

***ALL** All libraries
***ALLUSR** All user-defined libraries, plus libraries containing user data and having names starting with Q
***CURLIB** The job's current library
***LIBL** The library list
***USRLIBL** The user portion of the job's library list

Object type
INPUT; CHAR(10)
The types of objects to search for. You may either enter a specific object type, or a special value of *ALL. For a complete list of the available object types, see the *CL Reference*.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Format of the Generated Lists

The object list consists of:

- A user area
- A generic header
- An input parameter section
- A list data section

For details about the user area and generic header, see "User Space Format for List APIs" on page 2-7. For details about the other items, see the following sections. For a detailed description of each field in the information returned, see "Field Descriptions" on page 46-8.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name

Offset		Type	Field
Dec	Hex		
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	Object name specified
38	26	CHAR(10)	Object library name specified
48	30	CHAR(10)	Object type specified

OBJL0100 List Data Section: The following information is returned in the list data section of the OBJL0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 46-8.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name used
10	A	CHAR(10)	Object library name used
20	14	CHAR(10)	Object type used

OBJL0200 List Data Section: The following information is returned in the list data section of the OBJL0200 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 46-8.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0100 format
30	1E	CHAR(1)	Information status
31	1F	CHAR(10)	Extended object attribute
41	29	CHAR(50)	Text description
91	5B	CHAR(10)	User-defined attribute
101	65	CHAR(7)	Reserved

OBJL0300 List Data Section: The following information is returned in the list data section of the OBJL0300 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 46-8.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0200 format
108	6C	BINARY(4)	Auxiliary storage pool
112	70	CHAR(10)	Object owner
122	7A	CHAR(2)	Object domain
124	7C	CHAR(8)	Creation date and time
132	84	CHAR(8)	Change date and time
140	8C	CHAR(10)	Storage
150	96	CHAR(1)	Object compression status
151	97	CHAR(1)	Allow change by program
152	98	CHAR(1)	Changed by program
153	99	CHAR(10)	Object auditing value
163	A3	CHAR(9)	Reserved

OBJL0400 List Data Section: The following information is returned in the list data section of the OBJL0400 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 46-8.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0300 format
172	AC	CHAR(10)	Source file name
182	B6	CHAR(10)	Source file library name
192	C0	CHAR(10)	Source file member name
202	CA	CHAR(13)	Source file updated date and time
215	D7	CHAR(10)	Creator's user profile
225	E1	CHAR(8)	System where object was created
233	E9	CHAR(9)	System level
242	F2	CHAR(16)	Compiler
258	102	CHAR(8)	Object level
266	10A	CHAR(1)	User changed
267	10B	CHAR(16)	Licensed program
283	11B	CHAR(10)	Program temporary fix (PTF)
293	125	CHAR(10)	Authorized program analysis report (APAR)
303	12F	CHAR(21)	Reserved

OBJL0500 List Data Section: The following information is returned in the list data section of the OBJL0500 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 46-8.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0400 format
324	144	CHAR(8)	Object saved date and time
332	14C	CHAR(8)	Object restored date and time
340	154	BINARY(4)	Save size
344	158	BINARY(4)	Save size multiplier
348	15C	BINARY(4)	Save sequence number

Offset		Type	Field
Dec	Hex		
352	160	CHAR(10)	Save command
362	16A	CHAR(71)	Save volume ID
433	1B1	CHAR(10)	Save device
443	1BB	CHAR(10)	Save file name
453	1C5	CHAR(10)	Save file library name
463	1CF	CHAR(17)	Save label
480	1E0	CHAR(8)	Save active date and time
488	1E8	CHAR(44)	Reserved

OBJL0600 List Data Section: The following information is returned in the list data section of the OBJL0600 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0500 format
532	214	CHAR(8)	Last-used date and time
540	21C	CHAR(8)	Reset date and time
548	224	BINARY(4)	Days-used count
552	228	CHAR(1)	Usage information updated
553	229	CHAR(23)	Reserved

OBJL0700 List Data Section: The following information is returned in the list data section of the OBJL0700 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJL0600 format
576	240	BINARY(4)	Object size
580	244	BINARY(4)	Object size multiplier
584	248	CHAR(1)	Object overflowed ASP indicator
585	249	CHAR(3)	Reserved

Field Descriptions

Allow change by program. A 1-character variable that is used to return the allow change by program flag. A 1 is returned if the object can be changed with the Change Object Description (QLICOBJD) API. A 0 is returned if the object cannot be changed with the API.

Authorized program analysis report (APAR). The identifier of the authorized program analysis report (APAR) that caused this object to be replaced. The field is blank if the object did not change because of an APAR.

Auxiliary storage pool. The auxiliary storage pool ID. Valid values are:

1 System auxiliary storage pool
2–16 User auxiliary storage pool

Changed by program. A 1-character variable that is used to return the changed by program flag. A 1 is returned if the object has been changed with the QLICOBJD API. A 0 is returned if the object has not been changed by the API.

Change date and time. The time at which the object was last changed, in system time-stamp format.

Compiler. The licensed program identifier, version number, release level, and modification level of the compiler. The field has a ppppppVvvRrrMmm format where:

ppppppp The licensed program identifier.
Vvv The character V is followed by a 2-character version number.
Rrr The character R is followed by a 2-character release level.
Mmm The character M is followed by a 2-character modification level.

The field is blank if you do not compile the program.

Creation date and time. The time at which the object was created, in system time-stamp format.

Refer to the information about the Materialize System Object (MATSOBJ) machine interface (MI) instruction in the *MI Functional Reference* for information about the system time-stamp format.

Creator's user profile. The name of the user that created the object.

Days-used count. The number of days the object was used. If the object does not have a last-used date, the count is 0.

Extended object attribute. The extended attribute of the object, such as a program or file type. Extended attributes further describe the object. For example, an object type of *PGM may have a value of RPG (RPG program) or CLP (CL program), and an object type of *FILE may have a value of PF (physical file), LF (logical file), DSPF (display file), SAVF (save file), and so on.

Format name. The format of the returned output.

Information status. Whether or not the QUSLOBJ API returns the requested information for this object. Possible values are:

blank The requested information is returned. No errors occurred.
A No information is returned because the job that called the QUSLOBJ API does not have adequate authority to the object.

D The requested information is returned. However, the object is damaged and should be re-created as soon as possible.
L No information is returned because the object is locked.

If the value of this field is A or L, your application should not use the other fields for the object. Only the object name, library, and type fields contain accurate data.

Last-used date and time. The date and time at which the object was last used, in system time-stamp format. If the object has no last-used date, the field contains hexadecimal zeros.

Licensed program. The name, release level, and modification level of the licensed program if the retrieved object is part of a licensed program. The 7-character name starts in character position 1, the version number starts in position 8, the release level starts in position 11, and the modification level starts in position 14. The field is blank if the retrieved object is not a part of a licensed program.

Object auditing value. A 10-character variable that is used to return the type of auditing for an object. The valid values are:

**NONE* No auditing occurs for this object when it is read or changed regardless of the user who is accessing the object.
**USRPRF* Audit this object only if the user accessing the object is being audited. The user profile for the job is tested to determine if auditing should be done for this object. The user profile can specify if only change access is audited or if both read and change accesses are audited for this object.
**CHANGE* Audit all change access to this object by all users on the system.
**ALL* Audit all access to this object by all users on the system. All access is defined as a read or change operation.

Object compression status. Whether the object is compressed or decompressed. The status is returned in a 1-character variable with one of these values:

Y Compressed.
N Permanently decompressed and compressible.
X Permanently decompressed and *not* compressible.
T Temporarily decompressed.
F Saved with storage freed; compression status cannot be determined.

Temporarily decompressed objects exist in both decompressed and compressed form. *Permanently* decompressed objects exist in decompressed form only. The system handles some decompression automatically, depending on the type of object, the operation performed on it, and its frequency of use. For an overview of object compression and decompression, see the *CL Programmer's Guide*. For details about how to explicitly compress and decompress objects, see the entries for these commands in the *CL Reference*: Compress Object (CPROBJ), Decompress Object (DCPOBJ), and Reclaim Temporary Storage (RCLTMPSTG).

Object domain. The domain that contains the object. The value is *U if the object is in the user domain, or *S if the object is in the system domain.

Object level. The object control level for the created object.

Object library name specified. The name of the object library as specified in the call to the API.

Object library name used. The name of the library containing the object.

Object name specified. The name of the object as specified in the call to the API.

Object name used. The name of the object.

Object overflowed ASP indicator. The 1-character variable that returns the object overflowed auxiliary storage pool (ASP) indicator. The value is 1 if the object overflowed the ASP in which it resides; the value is 0 if the object has not overflowed the ASP. For objects in the system ASP, a 0 is always returned because it is not possible for an object that resides in the system ASP to overflow its ASP.

Object owner. The name of the object owner's user profile.

Object restored date and time. The time at which the object was restored, in system time-stamp format. If the object has never been restored, the field contains hexadecimal zeros.

Object saved date and time. The time at which the object was saved, in system time-stamp format. If the object has never been saved, the field contains hexadecimal zeros.

Object size. The size of the object in units of the size multiplier. The object size is equal to or smaller than the object size multiplied by the object size multiplier.

Object size multiplier. The value to multiply the object size by to get the true size. The value is 1 if the object is smaller than 1 000 000 000 bytes, and 1024 if it is larger.

Object type specified. The type of the object as specified in the call to the API.

Object type used. The type of the object. For a list of all the available object types, see the *CL Reference*.

Program temporary fix (PTF). The number of the program temporary fix (PTF) number that caused this object to be replaced. This field is blank if the object was not changed because of a PTF.

Reserved. An ignored field.

Reset date and time. The date the days-used count was last reset to zero, in system time-stamp format. If the days-

used count has never been reset, the field contains hexadecimal zeros.

Save active date and time. The date and time the object was last saved when the SAVACT(*LIB, *SYSDFN, or *YES) save operation was specified, in system time-stamp format. This parameter is found on the Save Library (SAVLIB), Save Object (SAVOBJ), Save Changed Object (SAVCHGOBJ), and Save Document Library Object (SAVDLO) CL commands. If the object has never been saved or if SAVACT(*NO) was specified on the last save operation for the object, the field contains hexadecimal zeros.

Save command. The command used to save the object. The field is blank if the object was not saved.

Save device. The type of device to which the object was last saved. The field is *SAVF if the last save operation was to a save file. The field is *DKT if the last save operation was to diskette. The field is *TAP if the last save operation was to tape. The field is blank if the object was not saved.

Save file library name. The name of the library that contains the save file if the object was saved to a save file. The field is blank if the object was not saved to a save file.

Save file name. The name of the save file if the object was saved to a save file. The field is blank if the object was not saved to a save file.

Save label. The file label used when the object was saved. The variable is blank if the object was not saved to tape or diskette. The value of the variable corresponds to the value specified for the LABEL parameter on the command used to save the object.

Save sequence number. The tape sequence number assigned when the object was saved on tape. The field contains zeros if the object was not saved.

Save size. The size of the object in bytes of storage at the time of the last save operation. The save size is equal to or smaller than the save size multiplied by the save size multiplier. The field contains zeros if the object was not saved.

Save size multiplier. The value to multiply the save size by to get the true size. The value is 1 if the save size is smaller than 1 000 000 000 bytes, and 1024 if it is larger.

Save volume ID. The tape or diskette volumes used for saving the object. The variable returns a maximum of ten 6-character volumes. The volume IDs begin in character positions 1, 8, 15, 22, 29, 36, 43, 50, 57, and 64. If more than 10 volumes are used, a 1 is returned in the 71st character of the variable; otherwise, the 71st character is blank. The field is blank if the object was last saved to a save file, or if it was never saved.

Source file library name. The name of the library that contains the source file used to create the object. The field is blank if no source file created the object.

Source file member name. The name of the member in the source file. The field is blank if no source file created the object.

Source file name. The name of the source file used to create the object. The field is blank if no source file created the object.

Source file updated date and time. The date and time the member in the source file was last updated. This field is in the CYYMMDDHHMMSS format where:

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

The field is blank if no source file created the object.

Storage. The storage status of the object data. *FREE indicates the object data is freed and the object is suspended. *KEEP indicates the object data is not freed and the object is not suspended.

System level. The level of the operating system when the object was created. The field has a VvvRrrMmm format where:

Vvv	The character V is followed by a 2-character version number.
Rrr	The character R is followed by a 2-character release level.
Mmm	The character M is followed by a 2-character modification level.

System where object was created. The name of the system on which the object was created.

Text description. The text description of the object. The field is blank if no text description is specified.

Usage information updated. Whether the object usage information is updated for this object type. The indicator is returned as Y (Yes) or N (No).

User changed. Whether the user program was changed. A character 1 is returned if the user changed the object. If the object was not changed by the user, the field is character 0.

User-defined attribute. Further defines an object type. This field is set by the user while using the QLICOBJD API.

User space library name. The library containing the user space, as specified in the call to the API.

User space name. The name of the user space.

Error Messages

CPF3CAA E List is too large for user space &1.

CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C20 E Error found by program &1.
CPD3C21 D Format name &1 is not valid.
CPD3C31 D Object type &1 is not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C31 E Object type &1 is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9800 E All CPF81xx messages could be signaled. xx is from 01 to FF.
CPF980F E Binding directory &1 in library &2 not found.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9804 E Object &2 in library &3 damaged.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9812 E File &1 in library &2 not found.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Rename Object (QLIRNMO) API

Parameters

Required Parameter Group:

1	From qualified object name	Input	Char(20)
2	Object type	Input	Char(10)
3	To qualified object name	Input	Char(20)
4	Replace object	Input	Char(1)
5	Error code	I/O	Char(*)

The Rename Object (QLIRNMO) API renames an existing object to a new object name or new library name or both, and optionally replaces the object.

When the replace object parameter requests to replace an existing object and the to object and library name already exist, the following occur:

- The *PUBLIC and private authorities from the existing object replace the authorities on the renamed object.
- The owner of the object is the owner of the from object.
- If the object is not a *PGM object, the existing object is deleted.
- For a *PGM object, the existing (to) object is moved to the QRPLOBJ library. The renamed program has the same user profile (USRPRF) and use adopted authority (USEADPAUT) values from a CRTxxxPGM CL command as the replaced program. If the existing program (to object) has the adopted authority USRPRF(*OWNER) attribute, the owner of the renamed program (from object) must be the owner of the existing program. An error message will be issued if the owners do not match. For more information on adopted authority, see the *Security Reference*.

All restrictions that apply to the Move Object (MOVOBJ) and Rename Object (RNMOBJ) commands also apply to the QLIRNMO API.

Neither this API nor the MOVOBJ command can move an object to the QTEMP library.

Authorities and Locks

Library Authority *CHANGE
Object Authority *OBJMGT

Note: Object types of *FILE, *JRN, *JRNRCV, and *MSGQ need *OBJOPR and *OBJMGT authorities.

Library Lock *EXCLRD
Object Lock *EXCL

If you replace an object, you must be the owner of the from object or have *ALLOBJ special authority. *ALLOBJ authority is needed to replace the authority on the from object.

When the request is to replace an existing object and the to object and library name parameter already exist, the following authority considerations apply:

- For *PGM objects, the user must have *OBJEXIST, *OBJMGT, and *READ authorities to the existing program object.
- For *MENU and *FILE objects, *OBJOPR and *OBJEXIST authorities are needed in addition to the other authorities listed.
- For other object types, *OBJEXIST authority is needed to delete the existing object in addition to the other authorities listed. *OBJEXIST and *USE authorities are needed to delete *LIB and *SBSD objects.

Required Parameter Group

From qualified object name
INPUT; CHAR(20)
The object being renamed and the library in which it is located. The first 10 characters contain the object name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The job's library list

Object type
INPUT; CHAR(10)
The type of object being renamed. If the from object and the to object belong to the same library, only a rename operation is done. The object type must be sup-

ported on the RNMOBJ command. If the from object and the to object belong to different libraries, a move operation is done. The object type must be supported on the MOVOBJ command. If both a rename and a move operation are done, the object type must be supported on both the RNMOBJ and MOVOBJ commands. An asterisk (*) must precede the object type. For a list of the object types that cannot be moved or renamed, see the *CL Programmer's Guide*.

To qualified object name
INPUT; CHAR(20)
The new name of the object and the new library in which it will be located. The object name can be the same name as the original object or a new name. The library name can be the same name as the original library or a new name. If the object name is the same as the original object, the library name must not be the same as the original library. The first 10 characters contain the object name, and the second 10 characters contain the library name.

Replace object
INPUT; CHAR(1)
Whether to replace an existing object with the same name as the to object and library name parameter. The following values can be specified:

0 Do not replace the existing object. If 0 is specified and the object already exists, an error message is returned to the application.
1 Replace the existing object.

Error code
I/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

CPF21A3 E &1 not valid for replace object option.
CPF21A4 E Objects cannot be moved into QTEMP.
CPF21A5 E Cannot replace object &1 in &2 type *&3.
CPF2111 E Library &1 already exists.
CPF2112 E Object &1 in &2 type *&3 already exists.
CPF2132 E Object &1 already exists in library &2.
CPF2136 E Renaming library &1 failed.
CPF2139 E Rename of library &1 failed.
CPF2140 E Rename of library &1 previously failed.
CPF2146 E Owner of new program and existing program not the same.
CPF2160 E Object type *&1 not eligible for requested function.
CPF2164 E Rename of library &2 not complete.
CPF2176 E Library &1 damaged.
CPF2183 E Object &1 cannot be moved into library &3.
CPF2189 E Not authorized to object &1 in &2 type *&3.
CPF2193 E Object &1 cannot be moved into library &4.
CPF22BC E Object &1 type &3 is not program defined.
CPF24B4 E Severe error while addressing parameter list.
CPF2512 E Operation not allowed for message queue &1.
CPF2691 E Rename of &2 type *&5 did not complete.
CPF2692 E Object &2 type *&5 must be varied off.
CPF2693 E &2 type *&5 cannot be used for rename.
CPF2694 E Object &2 type *&5 cannot be renamed.
CPF3CF1 E Error code parameter not valid.
CPF7010 E Object &1 in &2 type *&3 already exists.
CPF88C4 E Value &1 for new object is more than 8 characters.
CPF9801 E Object &2 in library &3 not found.
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.
CPF9809 E Library &1 cannot be accessed.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9812 E File &1 in library &2 not found.
CPF9814 E Device &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9821 E Not authorized to program &1 in library &2.
CPF9830 E Cannot assign library &1.
CPF9831 E Cannot assign device &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Object Description (QUSROBJD) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Object and library name	Input	Char(20)
5	Object type	Input	Char(10)

Optional Parameter:

6	Error code	I/O	Char(*)

The Retrieve Object Description (QUSROBJD) API lets you retrieve object information about a specific object. This information is similar to the information returned using the Display Object Description (DSPOBJD) command.

You can use the QUSROBJD API to:

- Determine who owns which objects in the specified libraries
- Provide disk management functions based on the object's size and use
- Provide backup analysis based on when the object was last saved or last updated

- Provide source member and object analysis from source member information to verify that the current source was used to create the specified object
- Work with a list of objects created by the QUSLOBJ API

Authorities and Locks

Library Authority *READ
Target Object Authority for Non-*FILE Objects
Any authority other than *EXCLUDE
Target Object Authority for *FILE Objects
*OBJOPR

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the requested information. It can be smaller than the format requested as long as the next parameter, length of receiver variable, specifies the length correctly. When this variable is smaller than the format, the API returns only the data that the variable can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. The minimum length is 8 bytes. Do not specify a length that is longer than the receiver variable; the results are unpredictable.

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

OBJD0100 Basic information (fastest)
OBJD0200 Information similar to that displayed by the programming development manager (PDM)
OBJD0300 Service information
OBJD0400 Full information (slowest)

These are described in the following sections.

Object and library name
INPUT; CHAR(20)
The object for which you want to retrieve information, and the library in which it is located. The first 10 characters contain the object name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Object type
INPUT; CHAR(10)
The type of object for which you want to retrieve the information. You can only specify external object types. Refer to the *CL Reference* for a complete list of available object types.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

OBJD0100 Format

The following information is returned for the OBJD0100 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 46-14.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Object name
18	12	CHAR(10)	Object library name
28	1C	CHAR(10)	Object type
38	26	CHAR(10)	Return library
48	30	BINARY(4)	Auxiliary storage pool
52	34	CHAR(10)	Object owner
62	3E	CHAR(2)	Object domain
64	40	CHAR(13)	Creation date and time
77	4D	CHAR(13)	Object change date and time

OBJD0200 Format

The following information is returned for the OBJD0200 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 46-14.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJD0100 format
90	5A	CHAR(10)	Extended object attribute
100	64	CHAR(50)	Text description
150	96	CHAR(10)	Source file name
160	A0	CHAR(10)	Source file library name
170	AA	CHAR(10)	Source file member name

OBJD0300 Format

The following information is returned for the OBJD0300 format. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 46-14.

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJD0200 format
180	B4	CHAR(13)	Source file updated date and time
193	C1	CHAR(13)	Object saved date and time
206	CE	CHAR(13)	Object restored date and time
219	DB	CHAR(10)	Creator's user profile
229	E5	CHAR(8)	System where object was created
237	ED	CHAR(7)	Reset date
244	F4	BINARY(4)	Save size
248	F8	BINARY(4)	Save sequence number
252	FC	CHAR(10)	Storage
262	106	CHAR(10)	Save command
272	110	CHAR(71)	Save volume ID
343	157	CHAR(10)	Save device
353	161	CHAR(10)	Save file name
363	16B	CHAR(10)	Save file library name
373	175	CHAR(17)	Save label
390	186	CHAR(9)	System level
399	18F	CHAR(16)	Compiler
415	19F	CHAR(8)	Object level
423	1A7	CHAR(1)	User changed
424	1A8	CHAR(16)	Licensed program
440	1B8	CHAR(10)	Program temporary fix (PTF)
450	1C2	CHAR(10)	Authorized program analysis report (APAR)

OBJD0400 Format

The following information is returned for the OBJD0400 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0		Everything from the OBJD0300 format
460	1CC	CHAR(7)	Last-used date
467	1D3	CHAR(1)	Usage information updated
468	1D4	BINARY(4)	Days-used count
472	1D8	BINARY(4)	Object size
476	1DC	BINARY(4)	Object size multiplier
480	1E0	CHAR(1)	Object compression status
481	1E1	CHAR(1)	Allow change by program
482	1E2	CHAR(1)	Changed by program
483	1E3	CHAR(10)	User-defined attribute
493	1ED	CHAR(1)	Object overflowed ASP indicator
494	1EE	CHAR(13)	Save active date and time
507	1FB	CHAR(10)	Object auditing value

Field Descriptions

Allow change by program. A 1-character variable that is used to return the allow change by program flag. A 1 is returned if the object can be changed with the Change Object Description (QLICOBJD) API. A 0 is returned if the object cannot be changed with the API.

Authorized program analysis report (APAR). The identifier of the authorized program analysis report (APAR) that caused this object to be replaced. The field is blank if the object did not change because of an APAR.

Auxiliary storage pool. The auxiliary storage pool ID.

1 System auxiliary storage pool
2–16 User auxiliary storage pool

Bytes available. The length of all data available to return. All available data is returned if enough space is provided.

Bytes returned. The length of the data actually returned.

Changed by program. A 1-character variable that is used to return the changed by program flag. A 1 is returned if the object has been changed with the QLICOBJD API. A 0 is returned if the object has not been changed by the API.

Compiler. The licensed program identifier, version number, release level, and modification level of the compiler. The field has a ppppppVvvRrrMmm format where:

ppppppp The licensed program identifier.
Vvv The character V is followed by a 2-character version number.
Rrr The character R is followed by a 2-character release level.
Mmm The character M is followed by a 2-character modification level.

The field is blank if you do not compile the program.

Creation date and time. The date and time the object was created. The creation date and time field is in the CYYMMDDHHMMSS format where:

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

HH Hour
MM Minute
SS Second

Creator's user profile. The name of the user that created the object.

Days-used count. The number of days the object was used. If the object does not have a last used date, the count is 0.

Extended object attribute. The extended attribute of the object, such as a program or file type. Extended attributes further describe the object. For example, an object type of *PGM may have a value of RPG (RPG program) or CLP (CL program), and an object type of *FILE may have a value of PF (physical file), LF (logical file), DSPF (display file), SAVF (save file), and so on.

Last-used date. The date the object was last used. This field is in the CYYMMDD format, which is the same format used for the reset date. If the object has no last-used date, the field is blank.

Licensed program. The name, release level, and modification level of the licensed program if the retrieved object is part of a licensed program. The 7-character name starts in character position 1, the version number starts in position 8, the release level starts in position 11, and the modification level starts in position 14. The field is blank if the retrieved object is not a part of a licensed program.

Object auditing value. A 10-character variable that is used to return the type of auditing for an object. The valid values are:

**NONE* No auditing occurs for this object when it is read or changed regardless of the user who is accessing the object.
**USRPRF* Audit this object only if the user accessing the object is being audited. The user profile for the job is tested to determine if auditing should be done for this object. The user profile can specify if only change access is audited or if both read and change accesses are audited for this object.
**CHANGE* Audit all change access to this object by all users on the system.
**ALL* Audit all access to this object by all users on the system. All access is defined as a read or change operation.

Object change date and time. The date and time the object was last changed. The format is the same as the creation date description, or it is blank if the object was not changed.

Object compression status. Whether the object is compressed or decompressed. The status is returned in a 1-character variable with one of these values:

Y Compressed.
N Permanently decompressed and compressible.
X Permanently decompressed and *not* compressible.
T Temporarily decompressed.
F Saved with storage freed; compression status cannot be determined.

Temporarily decompressed objects exist in both decompressed and compressed form. *Permanently* decompressed objects exist in decompressed form only. The system handles some decompression automatically, depending on the type of object, the operation performed on it, and its frequency of use. For an overview of object compression and decompression, see the *CL Programmer's Guide*. For details about how to explicitly compress and decompress objects, see the entries for these commands in the *CL Reference*: Compress Object (CPROBJ), Decompress Object (DCPOBJ), and Reclaim Temporary Storage (RCLTMPSTG).

Object domain. The domain that contains the object. The value is *U if the object is in the user domain, or *S if the object is in the system domain.

Object level. The object control level for the created object.

Object library name. The name of the library containing the object.

Object name. The name of the object.

Object overflowed ASP indicator. The 1-character variable that returns the object overflowed auxiliary storage pool (ASP) indicator. The value is 1 if the object overflowed the ASP in which it resides; the value is 0 if the object has not overflowed the ASP. For objects in the system ASP, a 0 is always returned because an object that resides in the system ASP cannot overflow its ASP.

Object owner. The name of the object owner's user profile.

Object restored date and time. The date and time the object was last restored. The format is the same as for the creation date, or it is blank if the object was never restored.

Object saved date and time. The date and time the object was last saved. The format is the same as for the creation date description, or it is blank if the object was never saved.

Object size. The size of the object in units of the size multiplier. The object size is equal to or smaller than the object size multiplied by the object size multiplier.

Object size multiplier. The value to multiply the object size by to get the true size. The value is 1 if the object is smaller than 1 000 000 000 bytes, and 1024 if it is larger.

Object type. The object type. For a list of all the available object types, see the *CL Reference*.

Program temporary fix (PTF). The number of the program temporary fix (PTF) number that caused this object to be replaced. This field is blank if the object was not changed because of a PTF.

Reset date. The date the days-used count was last reset to 0. The reset date field is in the CYYMMDD format, where:

C Century
YY Year
MM Month
DD Day

If the days-used count was not reset, the date is blank.

Return library. The name of the library containing the object if *LIBL or *CURLIB is specified for the library name on the object parameter.

Save active date and time. The date and time the object was last saved when the SAVACT(*LIB, *SYSDFN, or *YES) save operation was specified, in system time-stamp format. This parameter is found on the Save Library (SAVLIB), Save Object (SAVOBJ), Save Changed Object (SAVCHGOBJ), and Save Document Library Object (SAVDLO) CL commands. The format is the same as for the creation date description, or it is blank if the object was never saved or if SAVACT(*NO) was specified on the last save operation for the object.

Save command. The command used to save the object. The field is blank if the object was not saved.

Save device. The type of device to which the object was last saved. The field is *SAVF if the last save operation was to a save file. The field is *DKT if the last save operation was to diskette. The field is *TAP if the last save operation was to tape. The field is blank if the object was not saved.

Save file library name. The name of the library that contains the save file if the object was saved to a save file. The field is blank if the object was not saved to a save file.

Save file name. The name of the save file if the object was saved to a save file. The field is blank if the object was not saved to a save file.

Save label. The file label used when the object was saved. The variable is blank if the object was not saved to tape or diskette. The value of the variable corresponds to the value specified for the LABEL parameter on the command used to save the object.

Save sequence number. The tape sequence number assigned when the object was saved on tape. The field contains zeros if the object was not saved.

Save size. The size of the object in bytes of storage at the time of the last save operation. The field contains zeros if the object was not saved.

Save volume ID. The tape or diskette volumes used for saving the object. The variable returns a maximum of 10 six-character volumes. The volume IDs begin in character positions 1, 8, 15, 22, 29, 36, 43, 50, 57, and 64. If more than 10 volumes are used, a '1' is returned in the 71st character of the variable; otherwise, the 71st character is blank. The field is blank if the object was last saved to a save file, or if it was never saved.

Source file library name. The name of the library that contains the source file used to create the object. The field is blank if no source file created the object.

Source file member name. The name of the member in the source file. The field is blank if no source file created the object.

Source file name. The name of the source file used to create the object. The field is blank if no source file created the object.

Source file updated date and time. The date and time the member in the source file was last updated. The field is in the same format as the creation time and date. The field is blank if no source file created the object.

Storage. The storage status of the object data. *FREE indicates the object data is freed and the object is suspended. *KEEP indicates the object data is not freed and the object is not suspended.

System level. The level of the operating system when the object was created. The field has a VvvRrrMmm format where:

Vvv The character V is followed by a 2-character version number.
Rrr The character R is followed by a 2-character release level.
Mmm The character M is followed by a 2-character modification level.

System where object was created. The name of the system on which the object was created.

Text description. The text description of the object. The field is blank if no text description is specified.

Usage information updated. Whether the object usage information is updated for this object type. The indicator is returned as Y (Yes) or N (No).

User changed. Whether the user program was changed. A character 1 is returned if the user changed the object. If the object was not changed by the user, the field is character 0.

User-defined attribute. Further defines an object type. This field is set by the user while using the QLICOBJD API.

Error Messages

CPF2101 E Object type *&1 not valid.
CPF2115 E Object &1 in &2 type *&3 damaged.
CPF2150 E Object information function failed.
CPF2151 E Operation failed for &2 in &1 type *&3.
CPF2451 E Message queue &1 is allocated to another job.
CPF3CF1 E Error code parameter not valid.

CPF3C07 E Error occurred while retrieving information from object &1.
CPD3C20 D Error occurred with receiver variable specified.
CPD3C21 D Format name &1 is not valid.
CPD3C24 D Length of the receiver variable is not valid.
CPD3C31 D Object type &1 is not valid.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF36F7 E Message queue QSYSOPR is allocated to another job.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9812 E File &1 in library &2 not found.
CPF9814 E Device &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9821 E Not authorized to program &1 in library &2.
CPF9822 E Not authorized to file &1 in library &2.
CPF9825 E Not authorized to device &1.
CPF9830 E Cannot assign library &1.
CPF9831 E Cannot assign device &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 15. Office APIs

Chapter 47. Office APIs

The office APIs allow the user to perform customized and additional actions with office data. These are the office APIs:

Aid Spelling (QTWAIDSP) allows you to retrieve a list of correctly spelled words that are similar in spelling to the input word.

Change Office Program (QOGCHGOE) allows you to set or change the document handling and document conversion exit programs.

Check Spelling (QTWCHKSP) accepts a list of one or more words and returns the list with an indicator of whether each word is valid.

Control Office Services (QOCCTLOF) makes requests of office services and indicates that several office tasks will be processed.

Display Directory Panels (QOKDSPDP) allows you to use the Change Directory Information display interactively without using OfficeVision/400 administration to change directory information for OfficeVision/400 users.

Retrieve Office Programs (QOGRTVOE) allows you to retrieve program names and attributes for the current document handling and document conversion exit programs.

Aid Spelling (QTWAIDSP) 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	Input word	Input	Char(*)
5	Length of input word	Input	Binary(4)
6	Input dictionaries	Input	Char(*)
7	Length of input dictionaries	Input	Binary(4)
8	Output dictionaries	Output	Char(*)
9	Length of output dictionaries	Input	Binary(4)
10	Error code	I/O	Char(*)

The Aid Spelling (QTWAIDSP) API returns a list of candidate words that are similar in spelling to the input word. The input word is generally the misspelled word and the candidate words are correctly spelled words that are spelled similarly to the input word.

The QTWAIDSP API can be used to do the following:

- Return an indication if the input word was misspelled
- Return a list of correctly spelled words which are similar to the input word

To perform the function, a user can specify a maximum of eight IBM or user language dictionaries in the input dictionaries parameter.

Authorities and Locks

Dictionary Authority *USE
Dictionary Library Authority *USE
Dictionary Lock Authority *SHRNUP

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information about candidate words. You can specify the size of this area to be smaller than necessary to hold information about each word as long as you specify the length parameter correctly. The API returns only the data that the area can hold. The minimum size area is 8 bytes.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The content and format of the information returned in the receiver variable. You can use this format:

AIDW0100 Return all candidate words. See "AIDW0100 Format" on page 47-2.

Input word
INPUT; CHAR(*)
Locate candidate words that are spelled similarly to this word.

Length of input word
INPUT; BINARY(4)
The length of the input word.

Input dictionaries
INPUT; CHAR(*)
A structure containing a list of up to eight dictionaries that is used to locate candidate words. For the format of this parameter see, "Input Dictionaries Format" on page 47-5.

Length of input dictionaries
INPUT; BINARY(4)
The length of the input dictionaries parameter. The only valid value is 172 bytes.

Output dictionaries
OUTPUT; CHAR(*)
A structure containing a list of up to eight spelling aid dictionaries that were actually used to find candidate words. For the format of this parameter, see "Output Dictionaries Format" on page 47-5.

Length of output dictionaries
INPUT; BINARY(4)
The length of the output dictionaries parameter. The following are valid values:

0 No dictionaries are returned in the output dictionaries parameter.
>0 The length of space available in the output dictionaries parameter. If this length is larger than the actual size of the output dictionaries parameter, the results may not be predictable.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

AIDW0100 Format

The following information is returned in the receiver variable for the AIDW0100 format. For detailed descriptions of the fields, see "Field Descriptions" on page 47-2.

Offset			
Dec	**Hex**	**Type**	**Field**
Note: Fixed section.			
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of words returned
12	C	BINARY(4)	Number of words available
16	10	BINARY(4)	Offset to input word
20	14	BINARY(4)	Length of input word
24	18	CHAR(1)	Misspelled
25	19	CHAR(3)	Reserved
28	1C	BINARY(4)	Offset to first word information entry
32	20	BINARY(4)	Length of word information entry
36	24	BINARY(4)	Reserved
Note: The input word for which candidate words were found. The decimal and hexadecimal offsets are not defined. The input word is found by using the offset to input word field.			
		CHAR(*)	Input word
Note: Format of a word information entry. The following fields are repeated for each word returned. The decimal and hexadecimal offsets depend on the number of words returned. The first word information entry is found by using the offset to first word information entry in the fixed section. Each subsequent word information entry is found by adding the length of a word information entry to the offset of the previous word information entry.			
		BINARY(4)	Offset to candidate word
		BINARY(4)	Length of candidate word
		BINARY(4)	Output dictionary number
		CHAR(*)	Reserved
Note: The following field is repeated for each word returned. Each word is located by the offset to candidate word field and the length of the word is specified in the length of candidate word field.			
		CHAR(*)	Candidate word

Field Descriptions

Bytes available. The total length of all data available.

Bytes returned. The length of the data actually returned. If the data is truncated because the receiver variable is not large enough to hold all of the data available, the value will be less than the bytes available.

Candidate word. The candidate word that is spelled similar to the input word.

Input word. The word actually used to find candidate words.

Length of candidate word. The length of the candidate word in bytes.

Length of input word. The length of the input word in bytes.

Length of word information entry. The length of one item in the word information entry section.

Misspelled. Determines if the word is misspelled.

0 The word is spelled correctly.
1 The word is misspelled.

Number of words available. The number of candidate words that were found.

Number of words returned. The number of words actually returned in the receiver variable. If this number is smaller than the number of words available, then all of the words could not fit in the receiver variable.

Offset to candidate word. The byte offset, from the beginning of the receiver variable to the candidate word.

Offset to input word. The byte offset, from the beginning of the receiver variable to the input word.

Offset to first word information entry. The byte offset, from the beginning of the receiver variable to the first item in the word information entry section. If no candidate words are found, the value will be 0.

Output dictionary number. The index into the output dictionary list where the candidate word originated.

Reserved. An ignored field.

Error Messages

CPF24B4	Severe error while addressing parameter list.
CPF3CF1	Error code parameter not valid.
CPF3CF2	Error(s) occurred during running of &1 API.
CPF3C19	Error occurred with receiver variable specified.
CPF3C21	Format name &1 not valid.
CPF3C24	Length of the receiver variable is not valid.
CPF8751	Number of dictionaries not valid.
CPF8752	No valid dictionaries were found.
CPF8754	Length of input word not valid.
CPF8755	Length of input dictionaries not valid.
CPF8756	Length of output dictionaries not valid.
CPF8757	Input word &1 is not valid.
CPF9872	Program &1 in library &2 ended. Reason code &3.

Change Office Program (QOGCHGOE) API

Parameters

Required Parameter Group:

1	Document handling qualified exit program	Input	Char(20)
2	Document handling program supports mail flag	Input	Char(1)
3	Document conversion qualified exit program	Input	Char(20)
4	Error code	I/O	Char(*)

Optional Parameter Group:

5	Activate application enabler support	Input	Char(1)
6	Activate mail handling support	Input	Char(1)
7	Activate PCFILE identification processing support	Input	Char(1)

The Change Office Program (QOGCHGOE) API allows you to set or change the document handling and document conversion exit programs. The default value for these exit programs is *IBM, which indicates that OfficeVision/400 processing should be used for the request or conversion.

Authority

You must have security administrator (*SECADM) authority to call this program.

Required Parameter Group

Document handling qualified exit program
INPUT; CHAR(20)
The name of the program called when document editing and print requests are made.

The first 10 characters contain the program name, and the second 10 characters contain the name of the library where the program is located.

***IBM** OfficeVision/400 should process document requests. The library name should be blanks if *IBM is specified in the program name.

Document handling program supports mail flag
INPUT; CHAR(1)
This flag indicates whether the document handling program supports mail requests (that is, MAILVIEW, MAILEDIT, EDIT using Revise a Copy, MAILFWD, MAILREPLY, PRINT and PRINTOPTS from the Work with Mail display). If an application does not plan to handle these functions, set the value to 0. Setting the value to 0 prevents the unnecessary overhead of filing the mail items and calling the document handling program. If the document handling program is *IBM, set the value to 1.

0 Mail requests are not supported.
1 Mail requests are supported.

This indicates whether the program specified in the document handling exit program and library name parameter should be called for mail requests.

Document conversion qualified exit program
INPUT; CHAR(20)
The name of the program that is used to do document conversions when office services requires the document to be in a different data stream format.

The first 10 characters contain the program name, and the second 10 characters contain the name of the library where the program is located.

***IBM** Use the IBM-supplied document conversion exit program. The library name should be blank if *IBM is specified in the program name.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Optional Parameter Group

Activate application enabler support
INPUT; CHAR(1)
If an application does not plan to handle these functions, set the value to 0. If the application requires application enabler support, set the value to 1. Any other value does not change the current activation of application enabler support.

0 Application enabler support is inactive.
1 Application enabler support is active.

Activate mail handling support
INPUT; CHAR(1)
If an application does not plan to handle these functions,

set the value to 0. If the program requires mail handling support, set the value to 1. Any other value does not change the current activation of mail handling support.

0 Mail handling support is inactive.
1 Mail handling support is active.

Activate PCFILE identification processing support
INPUT; CHAR(1)
If an application does not plan to handle these functions, set the value to 0. If the program requires PCFILE identification processing support,[1] set the value to 1. Any other value does not change the current activation of PCFILE identification processing support.

0 PCFILE identification processing support is inactive.
1 PCFILE identification processing support is active.

Error Messages

CPF3C29 Object name &1 is not valid.
CPF90A8 *SECADM special authority required to do requested operation.
CPF9872 E Program &1 in library &2 ended. Reason code &3.
OFC15FF Value &1 for mail handled parameter is not valid.
OFC8019 Required module not on system.

Check Spelling (QTWCHKSP) 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	Word list	Input	Char(*)
5	Length of word list	Input	Binary(4)
6	Input dictionaries	Input	Char(*)
7	Length of input dictionaries	Input	Binary(4)
8	Output dictionaries	Output	Char(*)
9	Length of output dictionaries	Input	Binary(4)
10	Error code	I/O	Char(*)

The Check Spelling (QTWCHKSP) API accepts a list of one or more words in the word list parameter, and returns one word array entry in the receiver variable for every word with an indication if it is a valid word. To retrieve additional information about a specific word, see the "Aid Spelling (QTWAIDSP) API" on page 47-1.

The QTWCHKSP API can be used to do the following:

- Check the spelling of an individual word
- Find misspelled words in a character string

To perform the function, a user can specify a maximum of eight IBM or user language dictionaries in the input dictionaries parameter.

Authorities and Locks

Dictionary Authority *USE
Dictionary Library Authority *USE
Dictionary Lock Authority *SHRNUP

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information about each word in the word list. You can specify the size of this area to be smaller than necessary to hold information about each word as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold. The minimum size of this area is eight bytes.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The content and format of the information returned in the receiver variable. For more information on the format see "CHKW0100 and CHKW0200 Formats" on page 47-5. The following are possible format names:

CHKW0100
Word array entries for only the misspelled words should be returned in the receiver variable.
CHKW0200
Word array entries for all words in the word list should be returned.

Word list
INPUT; CHAR(*)
A list of one or more words to be checked. The words must be separated by one or more word separators where the blank character is always considered a word separator. The CCSID of the current job determines what code page will be used and each code page defines what other characters can be used as word separators. Refer to the *National Language Support Planning Guide* for more information on CCSIDs. Refer to "Word Separator Tables" on page 47-10 for information on word separators.

[1] PCFILE is a file assigned the document type of PCFILE (Document Interchange Architecture type of 14) by the PC Support/400 program.

Length of word list
INPUT; BINARY(4)
The length of the word list in bytes.

Input dictionaries
INPUT; CHAR(*)
A structure containing a list of up to eight dictionaries to be used to determine if the word or words in the word list are spelled correctly. For the format of this parameter, see "Input Dictionaries Format" on page 47-5.

Length of input dictionaries
INPUT; BINARY(4)
The length of the input dictionaries parameter. The only valid value is 172 bytes.

Output dictionaries
OUTPUT; CHAR(*)
A structure containing a list of up to eight dictionaries that were actually used. For the format of this parameter, see "Output Dictionaries Format" on page 47-5.

Length of output dictionaries
INPUT; BINARY(4)
The length of the output dictionaries parameter. The following are valid values:

0 No dictionaries are returned in the output dictionaries parameter.
>0 The length of space available in the output dictionaries parameter. If this length is larger than the actual size of the output dictionaries parameter, the results may not be predictable.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

CHKW0100 and CHKW0200 Formats

The following information is returned for both the CHKW0100 and CHKW0200 formats. If the receiver variable is large enough to hold all the information, then it contains information about each word in the word list. The receiver variable has three logical sections which contain:

- Fixed section
- Word information entry
- An array of the actual input words

For more detailed descriptions of the fields in the table, see "Field Descriptions" on page 47-6.

Offset		Type	Field
Dec	Hex		
Note: Fixed section.			
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Number of words returned
12	C	BINARY(4)	Number of words available
16	10	BINARY(4)	Offset to first word information entry
20	14	BINARY(4)	Length of word information entry
24	18	BINARY(4)	Reserved
Note: Format of a word information entry. The following four fields are repeated for each word returned. The first word information entry is found by using the offset to first word information entry in the fixed section. Each subsequent word information entry is found by adding the length of a word information entry to the offset of the previous word information entry.			
		BINARY(4)	Offset to word
		BINARY(4)	Length of word
		CHAR(1)	Misspelled
		CHAR(*)	Reserved
Note: The following field is repeated for each word returned. Each word is located by the offset to word field and the length of the word is specified in the length of word field.			
		CHAR(*)	Word

Input Dictionaries Format

The following list shows the format of the input dictionaries parameter. For detailed descriptions of the fields, see "Field Descriptions" on page 47-6.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Offset to dictionary entries
4	4	BINARY(4)	Number of dictionary entries
8	8	BINARY(4)	Reserved (must be set to 0)
Note: Format of a dictionary entry. The following fields are repeated by the number of dictionary entries. The decimal and hexadecimal offsets depend on the number of dictionary entries. The first dictionary entry is found by using the offset to dictionary entries in the header section.			
		CHAR(10)	Dictionary name
		CHAR(10)	Dictionary library name

Output Dictionaries Format

The following shows the format of the output dictionaries parameter. For detailed descriptions of the fields, see "Field Descriptions" on page 47-6.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of dictionaries returned

Offset			
Dec	Hex	Type	Field
4	4	BINARY(4)	Number of dictionaries available
Note: The following fields are repeated by the number of dictionaries returned. The decimal and hexadecimal offsets depend on the number of dictionaries returned.			
		CHAR(10)	Dictionary name
		CHAR(10)	Dictionary library name

Field Descriptions

Bytes available. The total length of all data available.

Bytes returned. The length of the data actually returned. If the receiver variable is not large enough to hold all of the data available, only the data that will fit in the space available is returned and this value will be less that the bytes available.

Dictionary library name. The library containing the language dictionary. The special values for the library name are *CURLIB and *LIBL.

Dictionary name. The name of the language dictionary. One special value for the name is *USERID.

**USERID* The name of the dictionary is the same as the user ID for the current job. A library name must be specified with this value.

Length of word. The length of the word in bytes.

Length of word information entry. The length of one item in the word information entry section.

Misspelled. Indicates if the word is misspelled.

0 The word is spelled correctly.
1 The word is misspelled.

Number of dictionaries available. The number of dictionaries that were actually used to determine if the word or words in the word list were spelled correctly.

Number of dictionaries returned. The number of dictionaries actually returned in the output dictionaries parameter. If this number is smaller than the number of dictionaries available, then all of the dictionaries could not fit in the space available.

Number of dictionary entries. The number of usable dictionaries to perform the function. Valid values are 1 through 8.

Number of words available. The number of words which are found in the word list.

Number of words returned. The number of words actually returned in the receiver variable. If this number is smaller than the number of words available, then all of the words could not fit in the receiver variable.

Offset to dictionary entries. The byte offset, from the beginning of the parameter, to the beginning of the dictionary entries.

Offset to first word information entry. The byte offset, from the beginning of the receiver variable, to the first item in the word information entry section.

Offset to word. The byte offset, from the beginning of the receiver variable, to the actual word.

Reserved. This field must be set to 0 on the input dictionaries parameter. Otherwise, this field is ignored.

Word. The word that was checked for spelling.

Error Messages

CPF24B4	Severe error while addressing parameter list.
CPF3CF1	Error code parameter not valid.
CPF3CF2	Error(s) occurred during running of &1 API.
CPF3C19	Error occurred with receiver variable specified.
CPF3C21	Format name &1 is not valid.
CPF3C24	Length of the receiver variable is not valid.
CPF8751	Number of dictionaries is not valid.
CPF8752	No valid dictionaries were found.
CPF8753	Length of word list not valid.
CPF8755	Length of input dictionaries not valid.
CPF8756	Length of output dictionaries not valid.
CPF9872	Program &1 in library &2 ended. Reason code &3.

Control Office Services (QOCCTLOF) API

Parameters

Required Parameter Group:

1	Request type	Input	Char(10)
2	Error code	I/O	Char(*)

The Control Office Services (QOCCTLOF) API makes requests of the office services. Office services accepts the following actions:

- *START
- *END
- *CHECK

Required Parameter Group

Request type
INPUT; CHAR(10)
The request that the application makes of the office services. The following are the possible values:

***START** Start an office services block. An office services block is the time during the running of an application when more office commands will be used. Office services leaves the job in such a state as to maximize the performance of subsequent office services use. This includes leaving the office files open and leaving working spaces created and already initialized. It is the responsibility of the application to end the office services block.

***END** End an office services block. No further office services will be used. The files will be closed and all working spaces deleted.

***CHECK** Determine if an office services block is active. An error will be returned if an office services block has not been previously started or has already ended.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

| CPF9872 E Program &1 in library &2 ended. Reason code &3.
OFCFD01 Office services session is already active.
OFCFD02 Office services session request failed.
OFCFD03 Office services session is not active.
OFCFD04 Office services session request is not correct.
OFCFD05 Number of bytes provided for the error code is not correct.

Display Directory Panels (QOKDSPDP) API

Parameters

Required Parameter Group:

1	User ID	Input	Char(8)
2	User address	Input	Char(8)
3	Title	Input	Char(10)
4	Function key processing	Output	Char(10)
5	Message to be displayed	Input	Char(*)
6	Error code	I/O	Char(*)

The Display Directory Panels (QOKDSPDP) API has the ability to display the Change Directory Information display. It can run that function interactively without using OfficeVision/400 administration to change directory information for office users. This API is for interactive use only.

Authority

If you have security administrator (*SECADM) authority, you can change all the directory information. If you do not have security administrator authority, then you can only change your own directory information.

Required Parameter Group

User ID
INPUT; CHAR(8)
The user ID to be processed. The ID needs to be in character set 697, code page 500, and monocase.

User address
INPUT; CHAR(8)
The address of the user to be processed. The address needs to be in character set 697, code page 500, and monocase.

Title
INPUT; CHAR(10)
The type of directory panel to be displayed. The following are the possible values:

***CHG** Change directory information
***ADD** Add directory information

Function key processing
OUTPUT; CHAR(10)
The operation processed for the key pressed. This indicates what type of ending processing to do. The following are the possible values:

***ENTER** The enter operation is processed.
***F3** The exit operation is processed.
***F12** The previous display is shown.

Message to be displayed
INPUT; CHAR(*)
A structure with a message ID and replacement text that is to be shown on the display. If the message data length is 0, no message will be displayed. For the format of the structure, see "Messages to Be Displayed 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.

Messages to Be Displayed Format

The format of the message to be displayed parameter is as follows:

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Message size

Offset			
Dec	Hex	Type	Field
4	4	CHAR(7)	Message name
11	0B	CHAR(*)	Message data

Field Descriptions

Message data. The data to be replaced in the message.

Message name. The message ID of the message to be displayed.

Message size. The length of the area that the calling application provides for the message data.

Error Messages

| CPF89A0 Values passed to QOKDSPDP are not valid.
| CPF9024 System cannot get correct record to finish operation.
| CPF9054 Description &1 already exists.
| CPF9083 User ID and address &1 &2 not changed.
| CPF9810 Library &1 not found.
| CPF9830 Cannot assign library &1.
| CPF9845 Error occurred while opening file &1.
| CPF9846 Error while processing file &1 in library &2.
| CPF9872 Program &1 in library &2 ended. Reason code &3.

Retrieve Office Programs (QOGRTVOE) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Error code	I/O	Char(*)

The Retrieve Office Programs (QOGRTVOE) API retrieves the programs that are used for office document requests and document conversion requests. The default value for the programs being retrieved is *IBM, which indicates that OfficeVision/400 processing should be used for the request or conversion.

Authority

To use this program you need *SECADM authority.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable to receive exit program information. This area must be large enough to accommodate the information specified.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. The length must be at least 8 bytes. If the variable is not large enough to hold the requested information, the data is truncated.

Format name
INPUT; CHAR(8)
The format of the program to be retrieved. The valid format is OGOE0100 (see "OGOE0100 Format").

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

OGOE0100 Format

The following list shows the format of the program to be retrieved. For detailed descriptions of the fields, see "Field Descriptions" on page 47-9.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Document handling program name
18	12	CHAR(10)	Document handling program library name
28	1C	CHAR(1)	Document handling program supports mail flag
29	1D	CHAR(10)	Document conversion program name
39	27	CHAR(10)	Document conversion program library name
49	31	CHAR(1)	Application enabler support active flag
50	32	CHAR(1)	Mail handling support active flag
51	33	CHAR(1)	PCFILE identification active flag
52	34	CHAR(1)	Application enabler support activated flag
53	35	CHAR(1)	Mail handling support activated flag
54	36	CHAR(1)	PCFILE identification activated flag
55	37	CHAR(1)	Application enabler support activating flag

Offset		Type	Field
Dec	Hex		
56	38	CHAR(1)	Mail handling support activating flag
57	39	CHAR(1)	PCFILE identification activating flag

Field Descriptions

Application enabler support active flag. An indication of whether the application enabler is active. The application enabler support is active if there is an application defined and the support has been made active.

0 The application enabler support will not be used with this system.
1 The application enabler support will be used with this system.

Application enabler support activated flag. An indication of whether the application enabler has been activated with the QOGCHGOE API.

0 The application enabler has not been activated.
1 The application enabler has been activated.

Application enabler support activating flag. An indication of whether the application enabler support can be activated. Application enabler support can become active if at least one application is defined and the support is made active with the QOGCHGOE API.

0 The application enabler would not become active if activated.
1 The application enabler would become active if activated.

Bytes available. The total length of all data available.

Bytes returned. The length of the data actually returned.

Document conversion program library name. The name of the library containing the program to be called for document conversions. This field will be blanks if the document conversion exit program name is *IBM.

Document conversion program name. The name of the program to be called for document conversions if an IBM conversion does not exist. If the program is *IBM, the IBM-supplied conversion exit or registered conversions will be called.

Document handling program library name. The name of the library containing the program to be called for document requests. This field will be blanks if the document handling exit program name is *IBM.

Document handling program name. The name of the program to be called for document requests. If the program is *IBM, OfficeVision/400 is called for the document requests.

Document handling program supports mail flag. An indication of whether the document handling program supports mail requests (that is, MAILVIEW, MAILFWD, MAILREPLY, PRINT and PRINTOPTS from the Work with Mail display). If *IBM is specified as the document handling exit program, the value will be 1.

0 The exit program does not support mail.
1 The exit program does support mail.

Mail handling support active flag. An indication of whether the Mail Handling support is active. Mail handling support is active if there is an application defined that handles mail, application enabler support has been activated, and mail support has been activated.

0 The mail handling support is inactive with this application.
1 The mail handling support is active with this application.

Mail handling support activated flag. An indication of whether the mail handling support could be activated. It is activated with the QOGCHGOE API.

0 The mail handling support has not been activated.
1 The mail handling support has been activated.

Mail handling support activating flag. An indication of whether the mail handling support could be activated. Mail handling support is activated with the QOGCHGOE API.

0 The mail handling support cannot be activated.
1 The mail handling support can be activated.

PCFILE identification active flag. An indication of whether the PCFILE identification processing support is active.[2] If PCFILE identification support has indicated that at least one type has been defined and PCFILE identification support is active, then the PCFILE identification processing is active.

0 The PCFILE identification processing support is inactive with this application.
1 The PCFILE identification processing support is active with this application.

PCFILE identification activated flag. An indication of whether the PCFILE identification processing support could be activated. PCFILE identification is activated with the Change Office Programs (QOGCHGOE) API.

0 The PCFILE identification support has not been activated.
1 The PCFILE identification support has been activated.

PCFILE identification activating flag. An indication of whether the PCFILE identification processing could be acti-

[2] PCFILE is a file assigned the document type of PCFILE (Document Interchange Architecture type of 14) by the PC Support/400 program.

vated. PCFILE identification support is activated with the QOGCHGOE API.

0 The PCFILE identification processing support would not become active if activated.

1 The PCFILE identification processing support would become active if activated.

Error Messages

CPF3C21 Format name &1 is not valid.
CPF3C24 Length of the receiver variable is not valid.
CPF90A8 *SECADM special authority required to do requested operation.
CPF9872 E Program &1 in library &2 ended. Reason code &3.
OFC8019 Required module not on system.

Word Separator Tables

The single-byte character set (SBCS) EBCDIC code page is extracted from the CCSID of the current job. The characters in the word list are mapped from the job's code page to the multinational code page 500 except for Greek and Turkish. Greek is mapped to code page 875; Turkish is mapped to code page 1026.

Delimiter Categories

Each character in a code page is assigned to a delimiter category (always, sometimes, and never a delimiter) as shown in the following tables.

Figure 47-1. Always Delimiters

Category	Context-Dependent Definition	Description	Examples (CP500)
A	—	Basic delimiter[1]	Blank
D	—	Other delimiters	{) &

[1]This category can only contain 1 character.

Figure 47-2. Never Delimiters

Category	Context-Dependent Definition	Description	Examples (CP500)
L	—	Lowercase alphabetic characters	a b c
U	—	Uppercase alphabetic characters	A B C

Figure 47-2. Never Delimiters

Category	Context-Dependent Definition	Description	Examples (CP500)
N	–	Numeric characters, not including superscripts or subscripts	0 1 2

Figure 47-3 (Page 1 of 2). Sometimes Delimiters

Category	Context-Dependent Definition	Description	Examples (CP500)
E	{ e | a < e < a }[1, 2]	Set of punctuation characters not treated as delimiters when surrounded by alphabetic characters	'
F	{ f | n < f < n }[3]	Set of punctuation characters not treated as delimiters when surrounded by numerics	. , : / -
G	{ g | p < g < n }[4]	Set of punctuation characters not treated as delimiters preceded by punctuation and followed by a numeric	. , $
H	{ h | ((p < h) and (h = p) }	Set of punctuation characters not treated as delimiters when immediately preceded by an identical character	. , : / - * = # %
I	{ i | F and H }	i is an element of F and H	. , : / -
J	{ j | E and F and H	j is an element of E and F and H	None

Figure 47-3 (Page 2 of 2). Sometimes Delimiters

Category	Context-Dependent Definition	Description	Examples (CP500)
K	{ k | F and G and H	k is an element of F and G and H	. ,

Note:

1. a = any character in L or U
2. < means precedes
3. h = any character in N
4. p = punctuation (nonalphabetic and nonnumeric)

Considerations for the Sometimes Delimiters Table

Categories E through K are usually called possible delimiters because they function as delimiters only in certain contexts.

Characters . (period), ! (exclamation point), and ? (question mark) have a special status. When identical characters from this category occur together in a sequence, the individual characters do not act as delimiters; the entire sequence of characters forms a single token. However, the sequence of characters taken together does act as a delimiter because the sequence forms a token separate from characters that precede and follow it. For example, the text streams:

- "abc...def" is broken into three tokens: "abc," "...," and "def."
- "Unbelievable!!!!" yields two tokens: "Unbelievable" and "!!!!."
- "Astonishing!!!???" yields three tokens: "Astonishing," "!!!," and "???."

A simple token table is a 256-element array of unsigned characters. The simple token category value for the character is found in the element indexed by the code point of each character. Each code point (character) must be assigned exactly one category. If, according to the above definition of sets, a character is a member of more than one category, it should be assigned to the highest level category (for example, the category with the letter name latest in alphabetical order).

The categories assigned to each character in the three code pages are shown in the following tables. See the *National Language Support Planning Guide* to refer to the characters that match these tables.

- Figure 47-4 shows the simple token table for code page 500.
- Figure 47-5 shows the simple token table for code page 875 (Greek).
- Figure 47-6 shows the simple token table for code page 1026 (Turkish).

Figure 47-4. Simple Token Table for Code Page 500

Hex Digits 1st → 2nd ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	A	D	I	L	U	D	D	D	D	D	D	N
-1	D	L	I	U	L	L	D	G	U	U	D	N
-2	L	L	U	U	L	L	L	G	U	U	U	N
-3	L	L	U	U	L	L	L	E	U	U	U	N
-4	L	L	U	U	L	L	L	D	U	U	U	N
-5	L	L	U	U	L	L	L	D	U	U	U	N
-6	L	L	U	U	L	L	L	D	U	U	U	N
-7	L	L	U	U	L	L	L	D	U	U	U	N
-8	L	L	U	U	L	L	L	D	U	U	U	N
-9	L	L	U	D	L	L	L	D	U	U	U	N
-A	D	D	D	I	D	D	D	D	D	D	D	D
-B	K	G	K	H	D	D	D	D	L	L	U	U
-C	D	H	H	D	L	L	U	D	L	L	U	U
-D	D	D	D	E	L	D	U	D	L	L	U	U
-E	D	D	D	H	L	U	U	E	L	L	U	U
-E	H	D	H	D	D	D	D	D	L	L	U	D

Figure 47-5. Simple Token Table for Code Page 875 Greek Support

Hex Digits 1st → 2nd ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	A	D	I	D	D	D	E	G	D	D	D	N
-1	U	U	I	U	L	L	D	L	U	U		N
-2	U	U	U	U	L	L	L	L	U	U	U	N
-3	U	U	U	U	L	L	L	L	U	U	U	N
-4	U	U	U	D	L	L	L	L	U	U	U	N
-5	U	U	U	U	L	L	L	L	U	U	U	N
-6	U	U	U	U	L	L	L	L	U	U	U	N
-7	U	U	U	U	L	L	L	L	U	U	U	N
-8	U	U	U	U	L	L	L	L	U	U	U	N
-9	U	U	U	D	L	L	L	L	U	U	U	N
-A	D	D	D	I	L	L	L	L	D	D	D	D
-B	K	G	K	H	L	L	L	L	L	D	D	D
-C	D	D	H	D	L	L	L	L	L			
-D	D	D	D	E	L	L	L	L	L	E		
-E	D	D	D	H	L	L	L	L	D	D	D	D
-F	H	D	H	D	L	L	L	L	D	D	D	D

Figure 47-6 (Page 1 of 2). Simple Token Table for Code Page 1026 Turkish Support

Hex Digits 1st → 2nd ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	A	D	I	L	U	D	L	D	L	D	L	N
-1	D	L	I	U	L	L	L	G	U	U	D	N
-2	L	L	U	U	L	L	L	G	U	U	U	N
-3	L	L	U	U	L	L	L	E	U	U	U	N
-4	L	L	U	U	L	L	L	D	U	U	U	N

Figure 47-6 (Page 2 of 2). Simple Token Table for Code Page 1026 Turkish Support

Hex Digits 1st → 2nd ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-5	L	L	U	U	L	L	L	D	U	U	U	N
-6	L	L	U	U	L	L	L	D	U	U	U	N
-7	L	L	U	U	L	L	L	D	U	U	U	N
-8	D	L	D	U	L	L	L	D	U	U	U	N
-9	L	L	U	D	L	L	L	D	U	U	U	N
-A	U	U	L	I	D	D	D	D	D	D	D	D
-B	K	U	K	U	D	D	D	D	L	L	U	U
-C	D	H	H	U	D	L	D	D	D	D	H	D
-D	D	D	D	E	G	D	G	D	L	L	U	U
-E	D	D	D	H	D	U	D	E	L	L	U	U
-E	H	H	H	U	D	D	D	D	L	L	U	D

Chapter 48. Office Exit Programs

You may write exit programs that are called by the operating system to customize the office functions to your needs.

Directory Search exit program allows the administrator to define a customized search of directory data. The Directory Search program is specified on the SCHPGM parameter of the Change Directory Attribute (CHGDIRA) command.

Directory Supplier exit program allows the administrator to decide whether a change, add, or delete operation for directory entries, departments, and locations is to be shadowed to collector systems. The supplier program is specified on the SUPPGM parameter of the Change Directory Attributes (CHGDIRA) command.

Directory Verification exit program allows the administrator to define additional security or validation checking when directory data is added, changed, or deleted. The verification program is specified on the VRFPGM parameter of the Change Directory Attribute (CHGDIRA) command.

Document Conversion exit program allows other document conversion programs to be called when a request is made for the OfficeVision/400 program to process a document that is an unsupported type (for example, PCFILE).

Document Handling exit program allows other editors and applications to be called from the OfficeVision/400 word processing and print functions.

Directory Search Exit Program

Parameters

Required Parameter Group:

1	Function being requested	Input	Char(10)
2	Maximum entries for addressees	Input	Binary(4)
3	Maximum entries for copy list	Input	Binary(4)
4	Number of array elements	Output	Binary(4)
5	Array	Output	Char(*)

The Directory Search exit program allows the administrator to define a customized search of directory data. The exit will perform one of three functions: display, select, or optional select (addressee or copy list selection). In this way, a user-customized search is provided and this search is integrated with OfficeVision/400 programs and the system distribution directory.

Use the Change Directory Attributes (CHGDIRA) command to specify the Directory Search exit program name in the SCHPGM parameter. When F10 is pressed on the Search System Directory display, the Directory Search exit program is called.

Required Parameter Group

Function being requested
INPUT; CHAR(10)
The type of search operation that was requested.

***DISPLAY**
Search and display directory information.

***SELECT**
Search and select user IDs. The output includes user IDs and addresses selected as a result of a search function.

***OPTSELECT**
Search and select user IDs for addressees and copy list.

Maximum entries for addressees
INPUT; BINARY(4)
The maximum number of entries that can be selected for the addressees of the distribution. The maximum number allowable is 100.

This parameter is used with the *SELECT and *OPTSELECT values of the function being requested parameter. If the function is *DISPLAY, this value is 0 indicating no output is returned.

Maximum entries for copy list
INPUT; BINARY(4)
The maximum number of entries that can be selected to be copied on the distribution. This is the maximum number of entries that can be returned in the array parameter with the value of 2 in select option used field. The maximum number allowable is 100.

This value is used with the *OPTSELECT value of the function being requested parameter. If the function is *DISPLAY, this value is 0 indicating no output is returned.

Number of array elements
OUTPUT; BINARY(4)
For the values *SELECT and *OPTSELECT, the number of users that is returned. The maximum value is the total of the maximum entries for addressees parameter plus the maximum entries for copy list parameter.

Array
OUTPUT; CHAR(*)
For the values *SELECT and *OPTSELECT, the array of the user IDs and addresses selected from the search function. For the format of this character parameter follows.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(1)	Select option used

Offset		Type	Field
Dec	Hex		
1	1	CHAR(8)	User ID
9	9	CHAR(8)	User address
17	11	CHAR(50)	User's full name or description

Field Descriptions

Select option used. The number of the option you want to use. The following are the possible values:

1 Addressee list
2 Carbon copy list (used only with *OPTSELECT)

User ID. The first part of a two-part network name used in the system distribution directory and in the office applications to uniquely identify a user.

User address. The second part of a two-part network name used in the system distribution directory and in the office applications to uniquely identify a user and to send electronic mail.

User's full name or description. The user's full name or any description up to 50 characters.

Directory Supplier Exit Program

Parameters

Required Parameter Group:

1	Function being requested	Input	Char(10)
2	Directory information format	Input	Char(10)
3	Owning system name	Input	Char(8)
4	User making request	Input	Char(10)
5	System making request	Input	Char(8)
6	Length of directory information	Input	Binary(4)
7	Directory information	Input	Char(*)
8	User exit program type	Input	Char(10)

The supplier program allows the administrator to decide whether operations (add, change, or delete) for directory entries, departments, and locations should be shadowed to collector systems. If a user exit program is defined, it is called before any changes are shadowed to a collector system. The supplier program is specified on the SUPPGM parameter of the Change Directory Attributes (CHGDIRA) command.

During directory shadowing, the supplier program is given all information known about the changes to the directory entries, departments, and locations. The exit program returns to the directory service an indication as to whether the add, change, or delete operation should be supplied to the collecting system.

The supplier program can reduce the amount of processing on the network, as opposed to the verification program. The supplier program can decide not to supply changes to the collecting system. However, the verification program can only decide not to apply changes that have already been supplied to the collecting system.

The supplier criteria should be consistent on all your systems to help reduce the amount of processing on the network.

Required Parameter Group

Function being requested
INPUT; CHAR(10)
The type of operation that the user is requesting to do to the directory information that is described by the other parameters.

***ADD** Information is being added.
***ADDDSC** User description is being added.
***CHG** Information is being changed.

All fields in the directory information that are not changed will be X'00' except for the first field of every table. That field indicates what directory entry, department, or location is being changed.

***DLT** Information is being deleted.
***DLTDSC** User description is being deleted.

Directory information format
INPUT; CHAR(10)
The format of the directory information that is being worked with. The information is provided in the directory information parameter. The valid formats are:

SUPP0100 Directory entry (See "SUPP0100 Format" on page 48-3.)
SUPP0200 Department entry (See "SUPP0200 Format" on page 48-4.)
SUPP0300 Location entry (See "SUPP0300 Format" on page 48-5.)

These formats have the same layout as the CHKP0100, CHKP0200, and CHKP0300 formats used for the Directory Verification exit program. This allows a single program to be used as both a supplier program and a verification program.

Owning system name
INPUT; CHAR(8)
The name of the system that "owns" the directory entry, department, or location that is being worked with. The owning system is the system that originally added the data to the network.

**LOCAL* The entry is owned by the local system.

User making request
INPUT; CHAR(10)
The user profile name of the user that is doing the request. When using the shadowing function, this is the user that originated the change.

System making request
INPUT; CHAR(8)
The system from which the request is coming. When using the shadowing function, this is the system that originated the change.

Length of directory information
INPUT; BINARY(4)
The length of the directory information in the directory information parameter. The length depends on the directory information format. Each format has a different (but fixed) length as shown in specific format tables.

Directory information
INPUT; CHAR(*)
The directory information that is associated with the directory entry, department, or location that the request is made against. For the format of this character parameter, refer to the specific format table ("SUPP0100 Format," "SUPP0200 Format" on page 48-4, or "SUPP0300 Format" on page 48-5) and to the "Field Descriptions" on page 48-5.

User exit program type
INPUT; CHAR(10)
The user exit program type that is associated with the call of the Directory Supplier exit program. This parameter is provided so that a single program can be used as both a supplier program and a verification program. This parameter is always set to *SUPPGM.

Rejecting a Directory Shadowing Record: The user-written exit program may reject requests to supply changes to a collector system. To do so, the exit program must signal a specific program message to the directory services module that called it, and then return. A directory shadowing operation is rejected based on the restrictions that have been identified. For example, if the exit program allows only five of ten systems in an advanced program-to-program communications (APPC) network to collect directory information, then directory shadowing requests by all other systems are rejected.

To allow a directory shadowing record to be supplied, the exit program only returns to the program that called it. Two program messages have been defined for the purpose of rejecting directory shadowing records. The messages are:

CPF89B6 Directory information not shadowed for authority reasons.
CPF89B8 Directory information not shadowed for data validation reasons.

You may provide an optional data structure with message variable substitution text. This will help clarify the reasons for the rejection to the users. The optional data structure is:

CHAR(10) The profile name of the user who requested directory shadowing. This information is from the user making request parameter.
CHAR(8) The system name of the user who requested directory shadowing. You can get this from the system making request parameter.
CHAR(120) A description of the reason the exit program is rejecting the request.

SUPP0100 Format

Offset		Type	Field
Dec	Hex		
Note: The following fields are in code page 500 and character set 697.			
0	0	CHAR(16)	User ID/address
16	10	CHAR(16)	System name/group
32	20	CHAR(10)	User profile
42	2A	CHAR(47)	Network user ID
89	59	CHAR(16)	New user ID/address
105	69	CHAR(16)	Old user to forward from user ID/address
121	79	CHAR(1)	Indirect user
122	7A	CHAR(1)	Print personal mail
123	7B	CHAR(3)	Reserved
Note: The character sets and code pages immediately follow the individual fields in the list below.			
126	7E	CHAR(50)	Description
176	B0	BINARY(4)	Character set
180	B4	BINARY(4)	Code page
184	B8	CHAR(40)	Last name
224	E0	BINARY(4)	Character set
228	E4	BINARY(4)	Code page
232	E8	CHAR(20)	First name
252	FC	BINARY(4)	Character set
256	100	BINARY(4)	Code page
260	104	CHAR(20)	Middle name
280	118	BINARY(4)	Character set
284	11C	BINARY(4)	Code page
288	120	CHAR(20)	Preferred name
308	134	BINARY(4)	Character set
312	138	BINARY(4)	Code page
316	13C	CHAR(2)	Reserved
318	13E	CHAR(50)	Full name
368	170	BINARY(4)	Character set
372	174	BINARY(4)	Code page
376	178	CHAR(2)	Reserved
378	17A	CHAR(10)	Department
388	184	BINARY(4)	Character set
392	188	BINARY(4)	Code page
396	18C	CHAR(2)	Reserved
398	18E	CHAR(50)	Job title
448	1C0	BINARY(4)	Character set

Offset		Type	Field
Dec	Hex		
452	1C4	BINARY(4)	Code page
456	1C8	CHAR(2)	Reserved
458	1CA	CHAR(50)	Company
508	1FC	BINARY(4)	Character set
512	200	BINARY(4)	Code page
516	204	CHAR(2)	Reserved
518	206	CHAR(26)	Telephone number 1
544	220	BINARY(4)	Character set
548	224	BINARY(4)	Code page
552	228	CHAR(2)	Reserved
554	22A	CHAR(26)	Telephone number 2
580	244	BINARY(4)	Character set
584	248	BINARY(4)	Code page
588	24C	CHAR(40)	Location
628	274	BINARY(4)	Character set
632	278	BINARY(4)	Code page
636	27C	CHAR(20)	Building
656	290	BINARY(4)	Character set
660	294	BINARY(4)	Code page
664	298	CHAR(16)	Office
680	2A8	BINARY(4)	Character set
684	2AC	BINARY(4)	Code page
688	2B0	CHAR(40)	Mailing address line 1
728	2D8	BINARY(4)	Character set
732	2DC	BINARY(4)	Code page
736	2E0	CHAR(40)	Mailing address line 2
776	308	BINARY(4)	Character set
780	30C	BINARY(4)	Code page
784	310	CHAR(40)	Mailing address line 3
824	338	BINARY(4)	Character set
828	33C	BINARY(4)	Code page
832	340	CHAR(40)	Mailing address line 4
872	368	BINARY(4)	Character set
876	36C	BINARY(4)	Code page
880	370	CHAR(2)	Reserved
882	372	CHAR(50)	Text
932	3A4	BINARY(4)	Character set
936	3A8	BINARY(4)	Code page
940	3AC	CHAR(1)	Print cover page
941	3AD	CHAR(1)	Mail notification
Note: The following X.400 fields are in the character set and code page as defined by 1984 X.400 standards.			
942	3AE	CHAR(3)	X.400 country
945	3B1	CHAR(16)	X.400 administration domain
961	3C1	CHAR(16)	X.400 private domain
977	3D1	CHAR(64)	X.400 organization
1041	411	CHAR(40)	X.400 surname
1081	439	CHAR(16)	X.400 given name
1097	449	CHAR(5)	X.400 initials
1102	44E	CHAR(3)	X.400 generation qualifier
1105	451	CHAR(32)	X.400 organization unit 1
1137	471	CHAR(32)	X.400 organization unit 2
1169	491	CHAR(32)	X.400 organization unit 3
1201	4B1	CHAR(32)	X.400 organization unit 4
1233	4D1	CHAR(8)	X.400 domain attribute type 1
1241	4D9	CHAR(128)	X.400 domain attribute value 1
1369	559	CHAR(8)	X.400 domain attribute type 2
1377	561	CHAR(128)	X.400 domain attribute value 2
1505	5E1	CHAR(8)	X.400 domain attribute type 3
1513	5E9	CHAR(128)	X.400 domain attribute value 3
1641	669	CHAR(8)	X.400 domain attribute type 4
1649	671	CHAR(128)	X.400 domain attribute value 4
1777	6F1	CHAR(3)	Reserved
1780	6F4	CHAR(32)	Fax telephone number
1812	714	BINARY(4)	Character set
1816	718	BINARY(4)	Code page
Note: All fields that are not changed will be X'00' except for the user ID/address field.			

SUPP0200 Format

Offset		Type	Field
Dec	Hex		
Note: The character sets and code pages immediately follow the individual fields in the list below.			
0	0	CHAR(2)	Reserved
2	2	CHAR(10)	Department
12	C	BINARY(4)	Character set
16	10	BINARY(4)	Code page
20	14	CHAR(2)	Reserved
22	16	CHAR(50)	Title

Offset		Type	Field
Dec	Hex		
72	48	BINARY(4)	Character set
76	4C	BINARY(4)	Code page
80	50	CHAR(2)	Reserved
82	52	CHAR(10)	Reports to department
92	5C	BINARY(4)	Character set
96	60	BINARY(4)	Code page
Note: The following field is in code page 500 and character set 697.			
100	64	CHAR(16)	Manager user ID/address
116	74	CHAR(2)	Reserved
118	76	CHAR(10)	Old department
128	80	BINARY(4)	Character set
132	84	BINARY(4)	Code page
Note: All fields that are not changed will be X'00' except for the department field and its corresponding character set and code page fields.			

SUPP0300 Format

Offset		Type	Field
Dec	Hex		
Note: The character sets and code pages immediately follow the individual fields in the list below.			
0	0	CHAR(40)	Location
40	28	BINARY(4)	Character set
44	2C	BINARY(4)	Code page
48	30	CHAR(2)	Reserved
50	32	CHAR(30)	Location line 1
80	50	BINARY(4)	Character set
84	54	BINARY(4)	Code page
88	58	CHAR(2)	Reserved
90	5A	CHAR(30)	Location line 2
120	78	BINARY(4)	Character set
124	7C	BINARY(4)	Code page
128	80	CHAR(2)	Reserved
130	82	CHAR(30)	Location line 3
160	A0	BINARY(4)	Character set
164	A4	BINARY(4)	Code page
168	A8	CHAR(2)	Reserved
170	AA	CHAR(30)	Location line 4
200	C8	BINARY(4)	Character set
204	CC	BINARY(4)	Code page
208	D0	CHAR(2)	Reserved
210	D2	CHAR(30)	Location line 5
240	F0	BINARY(4)	Character set
244	F4	BINARY(4)	Code page
248	F8	CHAR(2)	Reserved
250	FA	CHAR(30)	Location line 6
280	118	BINARY(4)	Character set
284	11C	BINARY(4)	Code page
288	120	CHAR(40)	Location changed to
328	148	BINARY(4)	Character set
332	14C	BINARY(4)	Code page
336	150	CHAR(40)	Old location
376	178	BINARY(4)	Character set
380	17C	BINARY(4)	Code page
Note: All fields that are not changed will be X'00' except for the location field and its corresponding character set and code page fields.			

Field Descriptions

Building. The name or number that identifies the user's building.

Character set. The character identifier (graphic character set) that was used by the work station to enter the data for the field.

Code page. The value specified on this parameter is used to instruct the printer device to interpret the hexadecimal byte string to print the same characters that were intended when the text was created.

Company. The name of the company for whom the user works.

Department. The name or number that identifies the user's department.

Description. The description associated with the user ID. One entry in the directory can have several different descriptions.

Fax telephone number. The facsimile telephone number.

First name. The user's first name or given name.

Full name. The user's full name as it appears when a directory is viewed or searched.

Indirect user. A user enrolled in the system distribution directory who receives mail but never signs on to view it. An indirect user receives printed mail only. The values are 0 for no and 1 for yes.

Job title. The title of the user's occupation.

Last name. The user's last name.

| **Location**. The location of the business or system. Some examples of location are city, state, or street address.

| **Location changed to**. The new location value after combining locations.

| **Location lines 1 through 6**. The location of the business or system. These fields further describe a location name. For example, the field may contain the general mailing address for the location.

| **Mail notification**. Whether or not the user wants to be notified when mail arrives.

| The user can specify these values:

| *Blank* Notified for priority or personal mail and messages
| *1* Notified for priority or personal mail and messages
| *2* Notified for only priority or personal mail
| *3* Notified for messages only
| *4* Notified for all mail
| *0* Notified for no mail

| **Mailing address lines 1 through 4**. The address of the user.

| **Manager user ID/address**. The department manager's user ID and address.

| **Middle name**. The user's middle name.

| **Network user ID**. A unique value associated with each user in the Enterprise Address Book. For example, the value could be the user ID/address, the social security number, or the employee number.

| **New user ID/address**. The new user ID and address used during the rename operation.

| **Office**. The name or number that identifies the user's office.

| **Old department**. The previous department value before being changed.

| **Old location**. The previous location value before being changed.

| **Old user to forward from user ID/address**. This field shows the previous user ID and address from which the user forwards mail.

| **Preferred name**. The name by which the user prefers to be known.

| **Print cover page**. Whether a cover page is printed when the user's mail is printed. The values are 0 for no and 1 for yes.

| **Print personal mail**. This field is used only if the user is an indirect user. The value specifies whether to print the mail that can be accessed only by the receiver, but not by someone working on behalf of the receiver. When mail is sent, it can be assigned the classification personal. The values are 0 for no and 1 for yes.

| **Reports to department**. The department to which this department reports.

| **Reserved**. An ignored field.

| **System name/group**. An IBM-supplied name that uniquely identifies the system. It is used as a network value for certain communications applications such as APPC.

| **Telephone number 1**. The telephone number of the user's office or business, or telephone numbers that are significant to the user. The most important number should be listed on the first line because only the first line is displayed when you use the search directory function.

| **Telephone number 2**. The second line for telephone numbers.

| **Text**. Any additional information that describes the entry.

| **Title**. A department title that further describes the department name.

| **User ID/address**. A two-part network name used in the system distribution directory and in the office applications to uniquely identify a user and to send electronic mail.

| **User profile**. The user profile name, if any, associated with a user ID and address.

| **X.400 administration domain**. The administration management domain part of the X.400 originator/recipient (O/R) name. An administration management domain is a management domain that is administered by a public organization, such as a national Post Telephone and Telegraph Administration (PTT). A management domain is a set of message transfer agents (MTAs) and user agents (UAs) that comprise a message handling system.

| **X.400 country**. The country part of the X.400 originator/recipient (O/R) name.

| **X.400 domain attribute types 1 through 4**. The type of a domain-defined attribute for this object. The domain-defined attribute is not defined by X.400 standards but is allowed in the X.400 originator/recipient (O/R) name to accommodate values of existing systems sending messages.

| **X.400 domain attribute values 1 through 4**. The code immediately following the attribute type that specifies a particular property from the set defined by the attribute type.

| **X.400 generation qualifier**. The generation qualifier part of the X.400 originator/recipient (O/R) name. For example, the generation qualifier in the name John R. Smith, III, is III. If you specify a generation qualifier, you must specify an X.400 surname.

| **X.400 given name**. The user first name, or given name, part of the X.400 originator/recipient (O/R) name. The

default for the given name is the equivalent of the first name. If you specify a given name, you must specify an X.400 surname.

X.400 initials. The first and middle initials of the X.400 originator/recipient (O/R) name. For example, the initials for John Henry Smith are JH. If you specify initials, you must specify a surname.

X.400 organization. The organization name part of the X.400 originator/recipient (O/R) name.

X.400 organization units 1 through 4. The organization-defined unit part of the X.400 originator/recipient (O/R) name.

X.400 private domain. The private management domain part of the X.400 originator/recipient (O/R) name. A private management domain is a management domain that is administered by a private company or a noncommercial organization.

X.400 surname. The user last name, or surname, part of the X.400 originator/recipient (O/R) name.

Error Messages

CPF89B6	Directory information not shadowed for authority reasons.
CPF89B8	Directory information not shadowed for data validation reasons.

Directory Verification Exit Program

Parameters

Required Parameter Group:

1	Function being requested	Input	Char(10)
2	Directory information format	Input	Char(10)
3	Owning system name	Input	Char(8)
4	User making request	Input	Char(10)
5	System making request	Input	Char(8)
6	Length of directory information	Input	Binary(4)
7	Directory information	Input	Char(*)
8	User exit program type	Input	Char(10)

The verification program allows the administrator to define additional security or syntax checking on the data. If defined, the user exit program is called before any directory entry, department, or location is added, changed, or removed from the system. The verification program is specified on the VRFPGM parameter of the Change Directory Attribute (CHGDIRA) command.

The verification program is given all updated information known about the directory entry, department, or location. The exit program returns to the directory service an indication as to whether the add, change, or delete operation is to be applied to the Enterprise Address Book. On the AS/400 system, this is called the system distribution directory. The EAB is a collection of data, such as information about people, departments, and locations in a network. An example of an enterprise is a company.

Whether or not update requests that are rejected by the exit program are supplied to other systems depends on the origin of the update.

- When a local update request is rejected by the exit program, the update does not affect the local system. If the AS/400 system is participating in directory shadowing, the update is not supplied to other systems in the directory shadowing network.
- If the AS/400 system is participating in directory shadowing and a shadowed update request is received and is rejected by the exit program, then the update does not affect the local system.

  Other systems in the directory shadowing network may be affected. The results depend on the nature of the shadowed update request:
  - When an add request is rejected, it is filtered out of (does not appear on) the local system, but remains on the network. Information about the add request is stored separately in a change log and will subsequently be supplied to other systems in the directory shadowing network.
  - When a change or delete request is rejected, the local system keeps its original information. In addition, the original information will be supplied again to other systems, including the system the information originated from, to keep directory information consistent throughout the network.

The verification criteria should be consistent on all your systems to help reduce the amount of processing on the network.

The system can get back the data which was filtered, but it is not a simple task. In order to retrieve the data which has been filtered out, the data can be shadowed again from the system where the data resides.

Required Parameter Group

Function being requested
INPUT; CHAR(10)
The type of operation that the user is requesting to do to the directory information that is described by the other parameters.

***ADD**	Information is being added.
***ADDDSC**	User description is being added.
***CHG**	Information is being changed. All fields in the directory information that are not changed will be X'00' except for the first field of every table. That field indicates what directory entry, department, or location is being changed.
***DLT**	Information is being deleted.

***DLTDSC** User description is being deleted.

Directory information format
INPUT; CHAR(10)
The format of the directory information that is being worked with. The information is provided in the directory information parameter. The valid formats are:

CHKP0100 Directory entry (See "CHKP0100 Format" on page 48-8.)
CHKP0200 Department entry (See "CHKP0200 Format" on page 48-10.)
CHKP0300 Location entry (See "CHKP0300 Format" on page 48-10.)

These formats have the same layout as the SUPP0100, SUPP0200, and SUPP0300 formats used for the Directory Supplier exit program. This allows a single program to be used as both a verification program and a supplier program.

Owning system name
INPUT; CHAR(8)
The name of the system that "owns" the directory entry, department, or location that is being worked with. The owning system is the system that originally added the data to the network.

**LOCAL* The entry is owned by the local system.

User making request
INPUT; CHAR(10)
The user profile name of the user that is doing the request. When using the shadowing function, this is the user that originated the modification.

System making request
INPUT; CHAR(8)
The system from which the request is coming. When using the shadowing function, this is the system that originated the modification.

Length of directory information
INPUT; BINARY(4)
The length of the directory information in the directory information parameter. The length depends on the directory information format. Each format has a different (but fixed) length as shown in specific format tables.

Directory information
INPUT; CHAR(*)
The directory information that is associated with the directory entry, department, or location that the request is made against. For the format of this character parameter, refer to the specific format table ("CHKP0100 Format","CHKP0200 Format" on page 48-10, or "CHKP0300 Format" on page 48-10) and to the "Field Descriptions" on page 48-11.

User exit program type
INPUT; CHAR(10)
The user exit program type that is associated with the call of the Directory Verification exit program. This parameter is provided so that a single program can be used as both a verification program and a supplier program. This parameter is always set to *VRFPGM.

Rejecting an Update Operation: The user-written exit program may reject update requests. To do so, the exit program must signal a specific program message to the directory services module that called it, and then return. An update is rejected based on the restrictions that have been identified. For example, if three people have authority to make updates but the program allows only one user to do updates, then update requests by all other users are rejected.

To allow a directory update request, the exit program only returns to the program that called it. Two program messages have been defined for the purpose of rejecting directory updates. The messages are:

CPF89A3 Operation not successful due to authority reasons.
CPF89A4 Operation not successful due to data validation reasons.

You may provide an optional data structure with message variable substitution text. This will help clarify the reasons for the rejection to the users. The optional data structure is:

CHAR(10) The profile name of the user who requested (entered) the update. This information is from the user making request parameter.
CHAR(8) The system name of the user who requested (entered) the update. You can get this from the system making request parameter.
CHAR(120) A description of the reason the exit program is rejecting the request.

CHKP0100 Format

Offset		Type	Field
Dec	Hex		
Note: The following fields are in code page 500 and character set 697.			
0	0	CHAR(16)	User ID/address
16	10	CHAR(16)	System name/group
32	20	CHAR(10)	User profile
42	2A	CHAR(47)	Network user ID
89	59	CHAR(16)	New user ID/address
105	69	CHAR(16)	Old user to forward from user ID/address
121	79	CHAR(1)	Indirect user
122	7A	CHAR(1)	Print personal mail
123	7B	CHAR(3)	Reserved
Note: The character sets and code pages immediately follow the individual fields in the list below.			
126	7E	CHAR(50)	Description
176	B0	BINARY(4)	Character set

Offset		Type	Field
Dec	Hex		
180	B4	BINARY(4)	Code page
184	B8	CHAR(40)	Last name
224	E0	BINARY(4)	Character set
228	E4	BINARY(4)	Code page
232	E8	CHAR(20)	First name
252	FC	BINARY(4)	Character set
256	100	BINARY(4)	Code page
260	104	CHAR(20)	Middle name
280	118	BINARY(4)	Character set
284	11C	BINARY(4)	Code page
288	120	CHAR(20)	Preferred name
308	134	BINARY(4)	Character set
312	138	BINARY(4)	Code page
316	13C	CHAR(2)	Reserved
318	13E	CHAR(50)	Full name
368	170	BINARY(4)	Character set
372	174	BINARY(4)	Code page
376	178	CHAR(2)	Reserved
378	17A	CHAR(10)	Department
388	184	BINARY(4)	Character set
392	188	BINARY(4)	Code page
396	18C	CHAR(2)	Reserved
398	18E	CHAR(50)	Job title
448	1C0	BINARY(4)	Character set
452	1C4	BINARY(4)	Code page
456	1C8	CHAR(2)	Reserved
458	1CA	CHAR(50)	Company
508	1FC	BINARY(4)	Character set
512	200	BINARY(4)	Code page
516	204	CHAR(2)	Reserved
518	206	CHAR(26)	Telephone number 1
544	220	BINARY(4)	Character set
548	224	BINARY(4)	Code page
552	228	CHAR(2)	Reserved
554	22A	CHAR(26)	Telephone number 2
580	244	BINARY(4)	Character set
584	248	BINARY(4)	Code page
588	24C	CHAR(40)	Location
628	274	BINARY(4)	Character set
632	278	BINARY(4)	Code page
636	27C	CHAR(20)	Building
656	290	BINARY(4)	Character set
660	294	BINARY(4)	Code page

Offset		Type	Field
Dec	Hex		
664	298	CHAR(16)	Office
680	2A8	BINARY(4)	Character set
684	2AC	BINARY(4)	Code page
688	2B0	CHAR(40)	Mailing address line 1
728	2D8	BINARY(4)	Character set
732	2DC	BINARY(4)	Code page
736	2E0	CHAR(40)	Mailing address line 2
776	308	BINARY(4)	Character set
780	30C	BINARY(4)	Code page
784	310	CHAR(40)	Mailing address line 3
824	338	BINARY(4)	Character set
828	33C	BINARY(4)	Code page
832	340	CHAR(40)	Mailing address line 4
872	368	BINARY(4)	Character set
876	36C	BINARY(4)	Code page
880	370	CHAR(2)	Reserved
882	372	CHAR(50)	Text
932	3A4	BINARY(4)	Character set
936	3A8	BINARY(4)	Code page
940	3AC	CHAR(1)	Print cover page
941	3AD	CHAR(1)	Mail notification
Note: The following X.400 fields are in the character set and code page as defined by 1984 X.400 standards.			
942	3AE	CHAR(3)	X.400 country
945	3B1	CHAR(16)	X.400 administration domain
961	3C1	CHAR(16)	X.400 private domain
977	3D1	CHAR(64)	X.400 organization
1041	411	CHAR(40)	X.400 surname
1081	439	CHAR(16)	X.400 given name
1097	449	CHAR(5)	X.400 initials
1102	44E	CHAR(3)	X.400 generation qualifier
1105	451	CHAR(32)	X.400 organization unit 1
1137	471	CHAR(32)	X.400 organization unit 2
1169	491	CHAR(32)	X.400 organization unit 3
1201	4B1	CHAR(32)	X.400 organization unit 4
1233	4D1	CHAR(8)	X.400 domain attribute type 1
1241	4D9	CHAR(128)	X.400 domain attribute value 1
1369	559	CHAR(8)	X.400 domain attribute type 2
1377	561	CHAR(128)	X.400 domain attribute value 2
1505	5E1	CHAR(8)	X.400 domain attribute type 3

Offset		Type	Field
Dec	Hex		
1513	5E9	CHAR(128)	X.400 domain attribute value 3
1641	669	CHAR(8)	X.400 domain attribute type 4
1649	671	CHAR(128)	X.400 domain attribute value 4
1777	6F1	CHAR(3)	Reserved
1780	6F4	CHAR(32)	Fax telephone number
1812	714	BINARY(4)	Character set
1816	718	BINARY(4)	Code page
Note: All fields that are not changed will be X'00' except for the user ID/address field.			

CHKP0200 Format

Offset		Type	Field
Dec	Hex		
Note: The character sets and code pages immediately follow the individual fields in the list below.			
0	0	CHAR(2)	Reserved
2	2	CHAR(10)	Department
12	C	BINARY(4)	Character set
16	10	BINARY(4)	Code page
20	14	CHAR(2)	Reserved
22	16	CHAR(50)	Title
72	48	BINARY(4)	Character set
76	4C	BINARY(4)	Code page
80	50	CHAR(2)	Reserved
82	52	CHAR(10)	Reports to department
92	5C	BINARY(4)	Character set
96	60	BINARY(4)	Code page
Note: The following field is in code page 500 and character set 697.			
100	64	CHAR(16)	Manager user ID/address
116	74	CHAR(2)	Reserved
118	76	CHAR(10)	Old department
128	80	BINARY(4)	Character set
132	84	BINARY(4)	Code page
Note: All fields that are not changed will be X'00' except for the department field and its corresponding character set and code page fields.			

CHKP0300 Format

Offset		Type	Field
Dec	Hex		
Note: The character sets and code pages immediately follow the individual fields in the list below.			
0	0	CHAR(40)	Location
40	28	BINARY(4)	Character set
44	2C	BINARY(4)	Code page
48	30	CHAR(2)	Reserved
50	32	CHAR(30)	Location line 1
80	50	BINARY(4)	Character set
84	54	BINARY(4)	Code page
88	58	CHAR(2)	Reserved
90	5A	CHAR(30)	Location line 2
120	78	BINARY(4)	Character set
124	7C	BINARY(4)	Code page
128	80	CHAR(2)	Reserved
130	82	CHAR(30)	Location line 3
160	A0	BINARY(4)	Character set
164	A4	BINARY(4)	Code page
168	A8	CHAR(2)	Reserved
170	AA	CHAR(30)	Location line 4
200	C8	BINARY(4)	Character set
204	CC	BINARY(4)	Code page
208	D0	CHAR(2)	Reserved
210	D2	CHAR(30)	Location line 5
240	F0	BINARY(4)	Character set
244	F4	BINARY(4)	Code page
248	F8	CHAR(2)	Reserved
250	FA	CHAR(30)	Location line 6
280	118	BINARY(4)	Character set
284	11C	BINARY(4)	Code page
288	120	CHAR(40)	Location changed to
328	148	BINARY(4)	Character set
332	14C	BINARY(4)	Code page
336	150	CHAR(40)	Old location
376	178	BINARY(4)	Character set
380	17C	BINARY(4)	Code page
Note: All fields that are not changed will be X'00' except for the location field and its corresponding character set and code page fields.			

Field Descriptions

Building. The name or number that identifies the user's building.

Character set. The character identifier (graphic character set) that was used by the work station to enter the data for the field.

Code page. The value specified on this parameter is used to instruct the printer device to interpret the hexadecimal byte string to print the same characters that were intended when the text was created.

Company. The name of the company for whom the user works.

Department. The name or number that identifies the user's department.

Description. The description associated with the user ID. One entry in the directory can have several different descriptions.

Fax telephone number. The facsimile telephone number.

First name. The user's first name or given name.

Full name. The user's full name as it appears when a directory is viewed or searched.

Indirect user. A user enrolled in the system distribution directory who receives mail but never signs on to view it. An indirect user receives printed mail only. The values are 0 for no and 1 for yes.

Job title. The title of the user's occupation.

Last name. The user's last name.

Location. The location of the business or system. Some examples of location are city, state, or street address.

Location changed to. The new location value after combining locations.

Location lines 1 through 6. The location of the business or system. These fields further describe a location name. For example, the field may contain the general mailing address for the location.

Mail notification. Whether or not the user wants to be notified when mail arrives.

The user can specify these values:

Blank	Notified for priority or personal mail and messages
1	Notified for priority or personal mail and messages
2	Notified for only priority or personal mail
3	Notified for messages only
4	Notified for all mail
0	No notification for mail

Mailing address lines 1 through 4. The address of the user.

Manager user ID/address. The department manager's user ID and address.

Middle name. The user's middle name.

Network user ID. A unique value associated with each user in the Enterprise Address Book. For example, the value could be the user ID/address, the social security number, or the employee number.

New user ID/address. The new user ID and address used during the rename operation.

Office. The name or number that identifies the user's office.

Old department. The previous department value before being changed.

Old location. The previous location value before being changed.

Old user to forward from user ID/address. This field shows the previous user ID and address from which the user forwards mail.

Preferred name. The name by which the user prefers to be known.

Print cover page. Whether a cover page is printed when the user's mail is printed. The values are 0 for no and 1 for yes.

Print personal mail. This field is used only if the user is an indirect user. The value specifies whether to print the mail that can be accessed only by the receiver, but not by someone working on behalf of the receiver. When mail is sent, it can be assigned the classification personal. The values are 0 for no and 1 for yes.

Reports to department. The department to which this department reports.

Reserved. An ignored field.

System name/group. An IBM-supplied name that uniquely identifies the system. It is used as a network value for certain communications applications such as APPC.

Telephone number 1. The telephone number of the user's office or business, or telephone numbers that are significant to the user. The most important number should be listed on the first line because only the first line is displayed when you use the search directory function.

Telephone number 2. The second line for telephone numbers.

Text. Any additional information that describes the entry.

Title. A department title that further describes the department name.

User ID/address. A two-part network name used in the system distribution directory and in the office applications to uniquely identify a user and to send electronic mail.

User profile. The user profile name, if any, associated with a user ID and address.

X.400 administration domain. The administration management domain part of the X.400 originator/recipient (O/R) name. An administration management domain is a management domain that is administered by a public organization, such as a national Post Telephone and Telegraph Administration (PTT). A management domain is a set of message transfer agents (MTAs) and user agents (UAs) that comprise a message handling system.

X.400 country. The country part of the X.400 originator/recipient (O/R) name.

X.400 domain attribute types 1 through 4. The type of a domain-defined attribute for this object. The domain-defined attribute is not defined by X.400 standards but is allowed in the X.400 originator/recipient (O/R) name to accommodate values of existing systems sending messages.

X.400 domain attribute values 1 through 4. The code immediately following the attribute type that specifies a particular property from the set defined by the attribute type.

X.400 generation qualifier. The generation qualifier part of the X.400 originator/recipient (O/R) name. For example, the generation qualifier in the name John R. Smith, III, is III. If you specify a generation qualifier, you must specify an X.400 surname.

X.400 given name. The user first name, or given name, part of the X.400 originator/recipient (O/R) name. The default for the given name is the equivalent of the first name. If you specify a given name, you must specify an X.400 surname.

X.400 initials. The first and middle initials of the X.400 originator/recipient (O/R) name. For example, the initials for John Henry Smith are JH. If you specify initials, you must specify a surname.

X.400 organization. The organization name part of the X.400 originator/recipient (O/R) name.

X.400 organization units 1 through 4. The organization-defined unit part of the X.400 originator/recipient (O/R) name.

X.400 private domain. The private management domain part of the X.400 originator/recipient (O/R) name. A private management domain is a management domain that is administered by a private company or a noncommercial organization.

X.400 surname. The user last name, or surname, part of the X.400 originator/recipient (O/R) name.

Error Messages

CPF89A3	Operation not successful due to authority reasons.
CPF89A4	Operation not successful due to data validation reasons.

Document Conversion Exit Program

Parameters

Required Parameter Group:

1	Input document name	Input	Char(12)
2	Input folder name	Input	Char(63)
3	Input document type	Input	Binary(4)
4	Output document name	Input	Char(12)
5	Output folder name	Input	Char(63)
6	Output document type	Input	Binary(4)
7	Function code	Input	Char(1)
8	Conversion existence indicator	Output	Char(1)

The Document Conversion exit program allows other document conversions to be called when a request is made for the OfficeVision/400 program to process a document type that it does not support. The OS/400 and OfficeVision/400 programs use document conversions when opening documents.

Required Parameter Group

Input document name
INPUT; CHAR(12)
The name of the document the function is to be performed against.

Input folder name
INPUT; CHAR(63)
The folder in which the document is to be found.

Input document type
INPUT; BINARY(4)
The type of the document that is being worked with. The value must be in the range of 1 through 65535.

Output document name
INPUT; CHAR(12)
The name of the output document.

Output folder name
INPUT; CHAR(63)
The folder in which the output document is to be placed.

Output document type
INPUT; BINARY(4)
The format of the document that is being worked with. The value must be in the range of 1 through 65535.

Function code
INPUT; CHAR(1)
Whether the exit is being called to check for the existence of a conversion or to perform the conversion.

0 A conversion is being requested.
1 An existence check is requested.

Conversion existence indicator
OUTPUT; CHAR(1)
Whether the requested conversion exists. This flag must be set for both conversion existence checks and conversion requests.

0 Conversion does not exist.
1 Conversion exists.

Use of Document Conversions by IBM Programs:

Other document conversions will be called when you are opening documents and:

- Processing the following document commands and using the OfficeVision/400 editor or print functions:
  - CRTDOC
  - EDTDOC
  - DSPDOC
  - PRTDOC
  - CHKDOC
  - MRGDOC
  - PAGDOC
  - ADDTXTIDXE
- Processing the following options from the Work with Documents in Folders display and using OfficeVision/400:
  - Create
  - Revise
  - View
  - Print
  - Spell
  - Paginate
  - Print options
- Processing the following options from the Work with Documents in a Document List display:
  - Revise
  - View
  - Print
- Processing the following options from the Work with Mail display and using OfficeVision/400:
  - Revise a copy
  - View
  - Print
  - Forward
  - Reply
- Copying a document on a create request when using the OfficeVision/400 editor.
- Recalling from or copying or moving to a notepad document within the OfficeVision/400 editor.
- Using the GET function within the OfficeVision/400 editor.
- Including a document with the .inc instruction in the OfficeVision/400 print function.
- Accessing external footnote documents in the OfficeVision/400 print function.
- Accessing a fill-in document (merge from a document) in the OfficeVision/400 print function.
- Accessing shell documents QPROFNOT and QPROFDOC for converting incoming PROFS* documents and notes.
- Printing mail for indirect office users.
- Accessing the shell note when creating a note.
- Accessing the output document for a copy graph or image request.

In each of these cases a conversion from the current format to RFTDCA or FFTDCA is requested. A second IBM conversion from DCA* to AS/400 format may also be requested using the IBM conversion programs.

Document Handling Exit Program

Parameters

Required Parameter Group:

1	Document name	Input	Char(12)
2	Folder name	Input	Char(63)
3	Document type	Input	Binary(4)
4	Function	Input	Char(26)
5	Function-specific information	Input	Char(485)
6	Exit processing indicator	Output	Char(4)

The Document Handling program allows other applications to be called in place of or in addition to the OfficeVision/400 word processor. Requests are sent to the exit program. The exit program can choose to process the request or return it to the OfficeVision/400 program for processing.

Required Parameter Group

Document name
INPUT; CHAR(12)
The name of the document on which the function is performed.

Folder name
INPUT; CHAR(63)
The folder in which the document is located.

Document type
INPUT; BINARY(4)
The format of the document that is being worked with. The value must be from 1 to 65535.

Function
INPUT; CHAR(26)
The first 10 characters is the type of operation that the user is requesting for this document. The next 16 characters contain the user ID and address of the current work on behalf of user. If there is no work on behalf of user, it will contain spaces.

Refer to the "DOCI0100 Format" on page 48-15 for print function requests.

Refer to the "DOCI0200 Format" on page 48-16 for merge function requests.

Refer to the "DOCI0300 Format" on page 48-17 for spell function requests.

Refer to the "DOCI0400 Format" on page 48-17 for mail function requests.

Refer to the "DOCI0500 Format" on page 48-17 for edit function requests.

Refer to the "DOCI0600 Format" on page 48-17 for create function requests.

Refer to the "DOCI0700 Format" on page 48-17 for fill form function requests.

Function-specific information

INPUT; CHAR(485)

Additional input that varies by function. See Figure 48-1 for more detailed information.

Note: Function-specific information that is not specified and for which there is no value stored in the document will be passed to the exit program as blanks.

Exit processing indicator

OUTPUT; CHAR(4)

The additional processing that the IBM programs should perform on return from the document handling program. Any code returned, when the function requested does not support it, will be treated as a return code of 0000.

0000 No additional processing required for this request. (Processed like the enter key from the exit.)

0001 Request has been processed, and the user requested to return. The IBM programs process as if F12 were pressed.

0002 Request has been processed, and the user requested to return. The IBM programs process as if F3 were pressed.

0007 Document has been deleted (only returned on VIEWMAIL requests).

0008 Request has been processed and the user requested that the item be sent. Only the MailEdit, MailForward, or MailReply functions are valid.

0010 The OfficeVision/400 program will process the requester as if the user exit had not been called. This indicator defines any additional processing the IBM programs perform when returning from the document handling program. Any code returned when the function requested does not support it, will be treated as a 0000 return.

0011 User is requesting that the document be filed locally. If any function other than MailView is used, it is treated as a 0000 return.

0012 User is requesting that the document be filed remotely. If any function other than MailView is used, it is treated as a 0000 return.

0013 User is requesting that the document be forwarded. If any function other than MailView is used, it is treated as a 0000 return.

0014 User is requesting a reply to the document. If any function other than MailView is used, it is treated as a 0000 return.

Figure 48-1 shows when the exit program will be called for each of the document functions.

Figure 48-1 (Page 1 of 2). Document Handling Function Requests

Function	Description	Where Called From
'CREATE'	Create document (DOCI0600 format)	• CRTDOC command • *Work with Documents in Folders* display; option 1 **Note:** The exit program is called after the Enter key or F10 is pressed on the Create Document Details display, when details are used.
'VIEW'	View document (No function specific format information)	• DSPDOC command • MERGE request with DSPOPT(*VIEW) • *Work with Documents in Folders* display; option 5 • *Work with Documents in a List* display; option 5
'EDIT'	Edit document and revise a copy of a mail item (DOCI0500 format)	• EDTDOC command • MERGE request with DSPOPT(*EDIT) • *Work with Documents in Folders* display; option 2 • *Work with Documents in a List* display; option 2 • *Work with Mail* display; option 2
'FILLFORM'	Fill Form Document (complete a form using the fill form editor of OfficeVision/400.) (DOCI0700 format)	• FILLFORM command • MRGDOC command; DSPOPT is *FILLFORM • *Work with Documents in a List* display; option 15 **Note:** The exit program is called after the Enter key or F10 is pressed on the Create Document Details display, when details are used.
'MAILVIEW'	View mail item (DOCI0400 format)	• *Work with Mail* display; option 5

Figure 48-1 (Page 2 of 2). Document Handling Function Requests

Function	Description	Where Called From
'MAILEDIT'	Create a note (DOCI0400 format)	• *OfficeVision/400* main menu; option 4 **Note:** The exit program is called after F6 (Type Note) is pressed on the *Send Note* display.
'MAILFWD'	Forward a note (DOCI0400 format)	• *Work with Mail* display; option 10 **Note:** The exit program is called after F6 (Type Note) is pressed on the *Forward Mail* display.
'MAILREPLY'	Reply to a note (DOCI0400 format)	• *Work with Mail* display; option 11 **Note:** The exit program is called after F6 (Type Note) is pressed on the *Reply to Mail* display.
'MERGE'	Merge document (DOCI0200 format)	• MRGDOC command; display merge options is *NO
'MERGEOPTS'	Merge document with options (DOCI0200 format)	• MRGDOC command; display merge options is *YES
'PRINT'	Print request (DOCI0100 format)	• PRTDOC command; display print options is *NO • *Work with Documents in Folders* display; option 6 • *Work with Documents in a List* display; option 6 • *Work with Mail* display; option 6
'PRINTOPTS'	Print with options (DOCI0100 format)	• PRTDOC command; Display print options is *YES • *Work with Documents in Folders* display; option 9 • *Work with Documents in a List* display; option 9 • *Work with Mail* display; option 9
'PAGINATE'	Paginate document (DOCI0100 format)	• PAGDOC command • *Work with Documents in Folders* display; option 13
'SPELLCHECK'	Check the spelling accuracy or grade level of a document (DOCI0300 format)	• CHKDOC command • *Work with Documents in Folders* display; option 11

DOCI0100 Format

This is the function-specific information for PRINT and PRINTOPTS requests. This format is filled out the same as it would have looked if the OfficeVision/400 program were to process the request. This format merges the parameters specified on the PRTDOC command with the print options record. The *Field* column describes the associated PRTDOC parameter for each format. The *Office Services Concepts and Programmer's Guide* has more information about the meaning of the parameters.

The "Field Descriptions" on page 48-17 have more detailed information about the fields for this format.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(2)	Number of copies to print
2	2	CHAR(1)	Output device
3	3	CHAR(10)	Printer device name
13	D	CHAR(10)	Output queue name
23	17	CHAR(10)	Output queue library name
33	21	CHAR(10)	Output file name
43	2B	CHAR(10)	Form type
53	35	CHAR(10)	Printer file name
63	3F	CHAR(10)	Printer file library name
73	49	CHAR(10)	Device file name
83	53	CHAR(10)	Device library name
93	5D	CHAR(10)	Device member name
103	67	CHAR(1)	Delay printing
104	68	CHAR(1)	Draft spacing
105	69	CHAR(1)	Print line numbers
106	6A	CHAR(1)	Resolve instructions
107	6B	CHAR(1)	Large print
108	6C	BINARY(4)	Graphic character set
112	70	BINARY(4)	Code page
116	74	CHAR(1)	Print separator page
117	75	CHAR(1)	Adjust line endings
118	76	CHAR(1)	Adjust page endings

Offset		Type	Field
Dec	Hex		
119	77	CHAR(1)	Allow widow lines
120	78	CHAR(2)	Additional spaces to left
122	7A	CHAR(1)	Print change symbols
123	7B	CHAR(5)	Change symbols to print
128	80	CHAR(1)	Print quality
129	81	CHAR(1)	Place on job queue
130	82	CHAR(1)	Send completion message
131	83	CHAR(10)	Job description name
141	8D	CHAR(10)	Job description library name
151	97	CHAR(1)	Cancel on error
152	98	CHAR(1)	Print error log
153	99	CHAR(10)	Error log form type
163	A3	CHAR(1)	Clear error log after printing
164	A4	CHAR(1)	Save resolved output
165	A5	CHAR(12)	Resolved output document
177	B1	CHAR(63)	Resolved output folder
240	F0	CHAR(1)	Multiple line report
241	F1	CHAR(1)	Print on both sides
242	F2	CHAR(1)	Automatic page binding
243	F3	CHAR(1)	Automatically shift margins to avoid truncating text
244	F4	CHAR(1)	Renumber system page numbers
245	F5	CHAR(7)	Print from page 1
252	FC	CHAR(8)	Print to page 1
260	104	CHAR(7)	Print from page 2
267	10B	CHAR(8)	Print to page 2
275	113	CHAR(7)	Print from page 3
282	11A	CHAR(8)	Print to page 3
290	122	CHAR(7)	Print from page 4
297	129	CHAR(8)	Print to page 4
305	131	CHAR(7)	Print from page 5
312	138	CHAR(8)	Print to page 5
320	140	CHAR(7)	Print from page 6
327	147	CHAR(8)	Print to page 6
335	14F	CHAR(7)	Print from page 7
342	156	CHAR(8)	Print to page 7
350	15E	CHAR(1)	Document is a label document
351	15F	CHAR(2)	Number of labels
353	161	CHAR(3)	Width of labels
356	164	CHAR(1)	Sheet-feed labels
357	165	CHAR(2)	Number of rows per sheet
359	167	CHAR(1)	Merge type
360	168	CHAR(10)	Query name
370	172	CHAR(10)	Query library name
380	17C	CHAR(12)	Data document name
392	188	CHAR(63)	Data folder name
455	1C7	CHAR(10)	Data file name
465	1D1	CHAR(10)	Data file library name
475	1DB	CHAR(10)	Data file member name

DOCI0200 Format

This is the structure that is used by this exit program when the function is for merge or merge with options. Note that the single-record merge is not supported.

The merge situations where the user specified DSPOPT(*VIEW) or DSPOPT(*EDIT) results in the exit program being called twice, once for the merge and then once for the subsequent view or edit. The *Field* column describes the associated MRGDOC parameter for each field. The *Office Services Concepts and Programmer's Guide* has more information about the meaning of the parameters.

The "Field Descriptions" on page 48-17 have more detailed information about the fields for this format.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(12)	To document name
12	C	CHAR(63)	To folder name
75	4B	CHAR(1)	Replace document
76	4C	CHAR(1)	Place on job queue
77	4D	CHAR(1)	Send completion message
78	4E	CHAR(10)	Job description name
88	58	CHAR(10)	Job description library name
98	62	CHAR(1)	Merge type
99	63	CHAR(10)	Query name
109	6D	CHAR(10)	Query library name
119	77	CHAR(12)	Data document name
131	83	CHAR(63)	Data folder name
194	C2	CHAR(10)	Data file name
204	CC	CHAR(10)	Data file library name
214	D6	CHAR(10)	Data file member name
224	E0	CHAR(1)	Adjustment option
225	E1	CHAR(1)	Multiple line report
226	E2	CHAR(1)	Collect footnotes
227	E3	CHAR(258)	Reserved

DOCI0300 Format

The "Field Descriptions" on page 48-17 explain the values to use for spell function requests.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Type of check
1	1	CHAR(7)	Beginning page number
8	8	CHAR(7)	Ending page number
15	F	CHAR(470)	Reserved

DOCI0400 Format

The "Field Descriptions" explain the values to use for mail function requests.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(12)	Mail reference document name
12	C	CHAR(63)	Mail reference folder name
75	4B	CHAR(1)	Attach mail reference
76	4C	CHAR(1)	Same note
77	4D	CHAR(408)	Reserved

DOCI0500 Format

The "Field Descriptions" explain the values to use for edit function requests.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Show exit display
1	1	CHAR(484)	Reserved

DOCI0600 Format

The "Field Descriptions" explain the values to use for create function requests.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Display exit display
1	1	CHAR(1)	Display document details display
2	2	CHAR(1)	Edit document
3	3	CHAR(482)	Reserved

DOCI0700 Format

The "Field Descriptions" explain the values to use for fill form function requests.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(12)	Shell form
12	C	CHAR(63)	Shell folder
75	4B	CHAR(12)	Save form
87	57	CHAR(63)	Save folder
150	96	CHAR(1)	Replace form
151	97	CHAR(1)	Display status
152	98	CHAR(1)	Display exit
153	99	CHAR(1)	Allow refresh
154	9A	CHAR(10)	Dump data file
164	A4	CHAR(10)	File library
174	AE	CHAR(10)	File member
184	B8	CHAR(1)	Replace or add
185	B9	CHAR(1)	Output data on exit
186	BA	CHAR(299)	Reserved

Field Descriptions

Additional spaces to left. The number of spaces added to the left margin in your printed document. If the document is not printed on both sides of the paper, this is useful when you want to bind your document. Valid values are 0 through 99.

Adjust line endings. The value Y (yes) adjusts line endings in the printed document. The lines are adjusted according to what is specified on the Line Spacing/Justification Options display. This is useful when you print a document that has data merged into it, has instructions, has display attributes that do not print as spaces, or uses a proportionally spaced font.

The value N (no) means you do not want to adjust the line endings in the printed document.

Adjust page endings. The value Y (yes) adjusts page endings in the printed document. The pages are determined by what is specified for the *First typing line* and *Last typing line* prompts on the Page Layout/Paper Options display.

The value N (no) means you do not want to adjust the page endings in the printed document.

Adjustment option. The values indicate the adjustment option you want to use.

1 None
2 Lines (Adjusts line endings in the printed document.)
3 Pages and lines (Adjusts page endings and lines in the printed document.)

Allow refresh. Whether the form refresh will be allowed from within the editor. The value 1 (yes) means the form refresh option will be allowed. The value 0 (no) means the refresh option will not be allowed.

Allow widow lines. The value Y (yes) means the page endings or column endings are determined by the exact number of lines per page as specified on the Page Layout/Paper Options display. Also use Y (yes) for the *Adjust page endings* prompt. When you allow widows, a single line from a paragraph could be separated from the rest of the paragraph.

The value N (no) means you do not want to have page endings or column endings determined by the exact number of lines.

Attach mail reference. Whether to concatenate the original mail item with the forward or reply operation. For MAILREPLY, this is entered by the user on the Reply to Mail display. For MAILFWD, this will always be set to a value of yes. The values are yes or no.

Automatic page binding. You can choose to adjust margins for page binding, which is a way to print pages and adjust margins for the even pages so that they will line up with the odd pages when printing on both sides. The values are yes or no.

Automatically shift margins to avoid truncating text. You can automatically shift the margin so that as much text as possible is printed if the margin exceeds the paper edge. The values are yes or no.

Beginning page number. A number from 1.0 through 9999.99 or *FIRST or *LAST to specify the page on which you want spell checking to start.

Cancel on error. The value Y (yes) means you want the document to stop printing if an error is detected during printing. The error is listed in the error log with an error message stating that the job is canceled.

The value N (no) means you want the document to print even if an error is detected during printing.

Change symbols to print. Different revision symbols result in a document when the revision symbol is changed. If your document contains more than one revision symbol character, and you do not select which revision symbol characters you want to print, all revision symbol characters specified in your document will print.

Clear error log after printing. The value Y (yes) means you want previous errors removed from the existing error log page before new ones are added. This is useful if you want the error log to contain only the errors that occurred during one printing of the document.

The value N (no) means you want your errors added to the end of the existing error log page. This is useful if you want to keep a history of all the times your document printed.

Code page. A particular assignment of hexadecimal identifiers to graphic characters. If you want to specify a code page, type the graphic character-set and code-page combination. A code page is a particular assignment of hexadecimal identifiers to graphic characters. When both the graphic character-set ID and code-page ID are typed, they must be separated by a hyphen.

Collect footnotes. The value Y (yes) means you want the associated footnote text for all Footnote instructions found in the included text to become part of the document created during the merge function. All Footnote instructions are changed to refer to the correct system page.

The value N (no) means you want the associated footnote text for all Footnote instructions found in the included text to remain outside of the document created during the merge function.

Data document name. The name of the document that contains the data to be merged.

The document name must be 1 to 12 characters in length.

Data file library name. The name of the library where the data file is located.

Data file member name. The name of the member that contains data.

Data file name. The name of the file to merge data from.

Data folder name. The name of the folder that contains the document that has the data to be merged.

Delay printing. The value Y (yes) delays printing your labels. The labels are held on the output queue where you can release them to print or delete them if you do not want them to print.

Device file name. The name of the file that contains the information about the device. Use the value 3 for file.

Device library name. A user-defined word that names a library. The value is *LIBL.

Device member name. The value *FILE for the member name, means the member with the same name as the file name will be used. The values are *FILE, *FIRST, and *LAST.

Display document details display. Whether or not you want the Document Details display shown. The values are Y (yes) or N (no).

Display exit. Whether or not the exit panel should be displayed.

0 No
1 Yes

Display exit display. Whether or not you want the exit display shown. The values are Y (yes) or N (no).

Display status. Whether or not status lines should be shown on the fill form display.

0 No
1 Yes

Document is a label document. The value (yes or no) identifies if the document is a label document. A label document contains the labels you are printing.

Draft spacing. The value Y (yes) doubles the spacing value in the *Line spacing* prompt on the Line Spacing/Justification Options display. For example, if the *Line spacing* prompt is 3 (Triple), the doubled spacing value is 6 and five blank lines are printed between each line of text in your document.

Dump data file. The name of the output file that will contain the form fill data.

Edit document. The values (yes or no) indicate whether or not you want to edit the document.

Ending page number. A number from 1.0 through 9999.99 or *FIRST or *LAST to specify the page on which you want spell checking to stop.

Error log form type. The forms type for the type of paper on which you want the error log to be printed. If you use the value *STD as the forms type, the error log will be printed on the paper specified in the printer file for the printer you selected. A printer file contains information controlling how your document is printed on a particular printer.

File library. The library to contain the dump data file.

File member. The member to contain the dump data. *FIRST is a special value indicating the first member of the file.

Form type. The host system form type for the forms control table (FCT) entry.

Graphic character set. The graphic character-set ID that you want to print your job. A graphic character-set ID is an identifier used to specify a set of graphic characters in a code page. This identifier can be two 4-digit prompts separated by a blank or by a hyphen.

Job description library name. The library name by specifying one of the following:

name The name of the library that is storing your job description.
**LIBL* If you use *LIBL, your library list is searched for the job description.

Job description name. The name of the job description that describes how the job is to be run.

Large print. The value Y (yes) prints your document with large print.

The value N (no) indicates that you do not want to print your document with large print.

Mail reference document name. The document name where a mail item is referred.

Mail reference folder name. The folder name where the referenced document is filed.

Merge type. The value identifies the way the documents are merged. The values are:

1 Query
2 Document
3 File and Blank

Multiple line report. The value Y (yes) means you want to create a multiple line report. A multiple line report merges Data Field instructions to create a report where each record of data produces several lines of output.

The value N (no) means you do not want to create a multiple line report.

Number of copies to print. The quantity of copies you want printed. The valid values are from 1 through 99.

Number of labels. The value that identifies the amount or quantity of labels on a page. The valid values are from 1 through 99.

Number of rows per sheet. If you selected Y (yes) for the *Sheet feed labels* prompt, this value from 1 through 99, is the number of rows of labels that you want printed on a page.

Output data on exit. Whether or not the output data is placed to an output file. The values are:

1 Yes
2 No

Output device. A device in a system by which data can be received from the system. The values are:

1 Printer
2 Display
3 File

Output file name. This is the name of the spooled file that is created. The values are *DOC and *FILE.

Output queue library name. The name of the library being used as the current library during the processing of this command. The value is *LIBL.

Output queue name. A list of output files to be printed or displayed. The output queue is used for spooled files. The values are *DEV, *FILE, and *WRKSTN.

Place on job queue. A document can be merged on the job queue.

Use the value Y (yes) if you want the document merged on the job queue. You can then continue working on your display while the document is being merged on the job queue.

Use the value N (no) if you want the job to merge interactively.

Print change symbols. The value Y (yes) prints revision symbols in the left margin of your document. Revision or change symbols are used to indicate lines that have been revised since the last time the document was changed. The revision symbol character is specified on the Change Editing Options display.

Use N (no) if you do not want to print the revision symbols in the left margin of your document.

Print error log. The value Y (yes) prints the error log page at the end of your document. The log contains any errors that were found while the document was being prepared for printing or errors found when a document was sent from another system. The log can also contain informational messages. If your document does not contain print errors, no log will be printed.

Print from pages 1 through 7. A number from 0.01 through 9999.99 to specify the page on which you want printing to start. Other valid values are *FIRST, *LAST, or *STRPAGE. If you use *FIRST, printing is started on the first page of the document. If you use *LAST, printing is started on the last page of the document. If you use *STRPAGE, the *To page* and *From page* values are the same and only one page is printed. The *From page* and *To page* prompts tell the printer to print specific pages from your document. The page values specified are the pages from the printed document.

Print line numbers. The value Y (yes) prints line numbers in your document. The line numbers begin with 1 on the first page of your document. Line numbers are not printed in headers or footers.

The value N (no) means you do not want to print line numbers in your document.

Print on both sides. The value indicates if you want your document printed on one side or both sides of a page. The values are:

1 No
2 Yes
3 Tumble

Print quality. The value indicates the quality of printing you want to use. The values are:

1 Letter

Letter-quality type is better print but causes wear on the printer ribbon (if your printer uses a ribbon) because the ribbon is struck harder.

2 Text

Text-quality type is not as good as letter-quality type, but is better than draft-quality type.

3 Draft

Draft-quality type saves wear on the printer ribbon (if your printer uses a ribbon) because the ribbon is not struck as hard as it would be if you were using letter-quality type or text-quality type.

Print separator page. A value that determines if sheets will separate the jobs in the printer. The value Y (yes) prints a separator page that includes such things as the document name, folder, document description, subject, reference, and authors. This is useful for separating your document from other documents.

Use N (no) if you do not want to print a separator page.

Print to pages 1 through 7. A number from 0.01 to 9999.99 is the page on which you want printing to stop. Other valid values are *FIRST and *LAST. If you use *FIRST, printing is ended on the first page of the document. If you use *LAST, printing is ended on the last page of the document. The *From page* and *To page* prompts tell the printer to print specific pages from your document. The page values specified are the pages from the printed document.

Printer device name. A device that writes output data from a system on paper or other media.

Printer file library name. The library name of the printer file. Possible values are:

name The name of the library in which the printer file is stored.
**LIBL* If you specify *LIBL, your library list is searched first for the printer file.

Printer file name. The name of the file where you want the printed labels to be received and stored. A file is a group of records that are related and treated as a unit. If the file does not exist, a new file will be created for you. If the file already exists, the document will be added to the end of the file.

Query library name. The name of the library that contains the query.

Query name. The name of the query that contains data to be merged.

Renumber system page numbers. The value Y (yes) renumbers the system page numbers.

The value N (no) keeps the same system page numbers.

Replace document. Whether the document replaces the existing document (yes) or is added to the existing documents (no).

Replace form. Whether or not the save form should be replaced if it already exists.

0 No
1 Yes

Replace or add. Whether the data is added or replaced in the file member.

A Add
R Replace

Reserved. An ignored field.

Resolve instructions. The value Y (yes) processes the instructions that you have placed in your document.

Resolved output document. The saved document is a resolved document (saved in printed form), that is, page endings are adjusted, line endings are adjusted, headers and footers are put in, and instructions are processed (optional). If you wish to print this document again, it takes less time to print because this document is already in printed form. (A resolved document is a document with the text instructions processed.) This document can be from the text editor or another system.

Resolved output folder. The name of the folder that will contain the document you specified in the *Document name* prompt. If the folder does not exist, a message is shown before it is created.

Same note. Whether the document handling program has previously been called to process this forward or reply operation. This indication provides a means for the document handling program to distinguish between the initial and subsequent calls to process this document. This can be used to avoid attaching the mail reference multiple times. The valid values are Y (yes) or N (no).

Save folder. The folder name where the save form document is saved.

Save form. The name of the document to save the results of the fill form process.

Save resolved output. The value Y (yes) means the document you are printing is saved in the document specified in the document prompt.

The value N (no) means you do not want the document you are printing saved in another document.

Send completion message. The value Y (yes) means you are putting your print job on the job queue and you want a message sent to your display station when the job has completed.

The value N (no) means you are putting your print job on the job queue and you do not want a message sent to your display station when the job has completed.

Shell folder. The folder containing the shell form.

Shell form. The shell document for fill form.

Sheet-feed labels. A value that defines that a device is attached to a printer to automatically feed out sheets of labels. The value Y (yes) means you are sheet-feed printing and want more than one row of labels on a page. If you are using sheet-feed paper, there is no other way to print more than one row of labels on a page.

Show exit display. Whether or not you want the exit display shown. The valid values are Y (yes) or N (no).

To document name. The name of the document to be merged with.

To folder name. The name of the folder that will contain the document being created.

Type of check. The value indicates if you want to check the spelling or the grade level.

0 Spell
1 Grade

Width of labels. The width of a label is the number of characters from the left edge of the first label to the left edge of the next label, including the blank spaces between the labels. If the width you specify is larger than the margins for your document, the margins are used as the width. Valid values are 2 through 198.

Note: For the printer device field the special values *SYSVAL, *USRPRF, and *WRKSTN are replaced with the appropriate printer name, therefore the exit program does not use the special values.

Error Messages

OFC1680	Message returned from document handling program.
OFC1690	OfficeVision/400 processing requested by program &1 in library &2.
OFC1691	Error while calling program &1 in library &2.

Part 16. Operational Assistant APIs

Chapter 49. Operational Assistant APIs

Most functions on the AS/400 Operational Assistant* menu can be accessed individually by calling APIs found in the QSYS library. These Operational Assistant APIs allow you to incorporate Operational Assistant functions into your application menus.

Your assistance level setting affects the type of display you see when these APIs are called.

The APIs in this chapter are presented in alphabetical order.

The Operational Assistant APIs are:

| **List Signed-On Users** (QEZLSGNU) generates a list of signed-on users and places the list in the specified user space.
Operational Assistant Attention-Key-Handling (group jobs) (QEZMAIN) creates a group job to display the AS/400 Operational Assistant menu.
Operational Assistant Attention-Key-Handling (nongroup jobs) (QEZAST) uses the GO ASSIST command to display the AS/400 Operational Assistant menu.
Save Information (QEZSAVIN) displays the Save Information to Help Resolve a Problem display.
| **Send Message** (QEZSNDMG) sends a message to one or more users or display stations and optionally shows the Operational Assistant Send a Message display before sending the message.
Work with Jobs (QEZBCHJB) displays either the Work with Jobs panel or the Work with User Jobs panel.
Work with Messages (QEZMSG) displays either the Work with Messages panel or the Display Messages panel.
Work with Printer Output (QEZOUTPT) displays either the Work with Printer Output panel or the Work with All Spooled Files panel.

List Signed-On Users (QEZLSGNU) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	User name	Input	Char(10)
4	Display station name	Input	Char(10)
5	Include disconnected jobs and suspended group jobs	Input	Char(10)
6	Include signed-off users with output waiting to print	Input	Char(10)
7	Error code	I/O	Char(*)

The List Signed-On Users (QEZLSGNU) API generates a list of signed-on users and places the list in the specified user space. The generated list replaces any existing lists in the user space.

When you specify a generic user name or a generic display station name, you can generate a subset of the signed-on user list. If both a user name and a display station name are specified, only entries that match both the user name and the display station name are included in the list of signed-on users.

You can use the QEZLSGNU API to get a list of users similar to that seen by using the Work with User Jobs (WRKUSRJOB) command with options STATUS (*ACTIVE) and JOBTYPE (*INTERACT).

Authorities and Locks

User Space Authority *CHANGE
Library Authority *USE
User Space Lock *EXCLRD

Required Parameter Group

Qualified user space 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:

SGNU0100 This format returns information about the user's job and what kind of activity they are performing.
SGNU0200 This format returns the same information as the SGNU0100 format, plus the text descriptions of the user profile and display station. This format requires more processing than the SGNU0100 format.

For more information, see "SGNU0100 Format" on page 49-2 or "SGNU0200 Format" on page 49-2.

User name
INPUT; CHAR(10)
A specific user name, a generic user name, or the following special value:

***ALL** All signed-on users

Display station name
INPUT; CHAR(10)
A specific display station name, a generic display station name, or the following special value:

***ALL** All display stations with signed-on users

Include disconnected jobs and suspended group jobs
INPUT; CHAR(10)
An indicator for whether or not to include disconnected jobs and suspended group jobs.

You must use one of the following special values:

***YES** Include signed-on users with disconnected jobs and users whose group jobs have been suspended.
***NO** Do not include signed-on users with disconnected jobs and users whose group jobs have been suspended.

Include signed-off users with output waiting to print
INPUT; CHAR(10)
An indicator for whether or not to include signed-off users with output waiting to print. This would include all interactive signed-off users with output waiting to print and all batch graphical user interface jobs that have ended with output waiting to print.

You must use one of the following special values:

***YES** Include signed-off users with output waiting to print.
***NO** Do not include signed-off users with output waiting to print.

Error code
I/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 signed-on user list consists of:

- A user area
- A generic header
- An input parameter section
- A list data section:
  - SGNU0100 format
  - SGNU0200 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."

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)	User name specified
38	26	CHAR(10)	Display station specified
48	30	CHAR(10)	Include disconnected jobs and suspended group jobs indicator specified
58	3A	CHAR(10)	Include signed-off users with output waiting to print indicator specified

SGNU0100 Format

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Display station name
10	A	CHAR(10)	User name
20	14	CHAR(6)	Job number
26	1A	CHAR(10)	Activity
36	24	CHAR(10)	Activity name
46	2E	CHAR(1)	Disconnect job allowed indicator
47	2F	CHAR(17)	Reserved

SGNU0200 Format

Offset			
Dec	Hex	Type	Field
0	0		Everything from the SGNU0100 format
64	40	CHAR(50)	Display station description
114	72	CHAR(50)	User description

Field Descriptions

Activity. Information about what is happening with the interactive job. Valid values are:

**BLDIDX* The user is using a file that is currently being rebuilt. The user's interactive job will be delayed until the file rebuild operation is complete or the Transfer to Secondary Job (TFRSECJOB) command has been run. The file name is in the activity name field.
**CMD* The user is running a command. The command name is in the activity name field.

| *CMDENT The user is currently on the Command Entry display.
| *DLY The Delay Job (DLYJOB) command has been run, and the user's interactive job will be delayed until the time limit specified has passed.
| *DSCJOB The user is temporarily signed-off (disconnected) the system (for example, option 80 on the ASSIST menu).
| *DUMP The user encountered an unexpected error, and a printout of diagnostic information (dump) is being created.
| *GRP The job is a suspended group job. The group name is in the activity name field.
| *HLD The user's interactive job is temporarily stopped and cannot run until it is released.
| *MNU The user is using a menu. The menu name is in the activity name field.
| *MSG The user is waiting for a reply to a message.
| *PGM The user is running a program. The program name is in the activity name field.
| *PRT The job has ended but has output waiting to print.
| *SIGNOFF A job is ending because the user has signed off the system, or the interactive job has ended due to one of the following:
  - End Job (ENDJOB) command
  - End Subsystem (ENDSBS) command
  - End Group Job (ENDGRPJOB) command
  - An unexpected error
| *SYSRQS The user has pressed the System Request key.
| *S36PRC The user is running a System/36 environment procedure. The procedure name is in the activity name field.

| **Activity name**. The name of the program, menu, command, System/36 procedure, or file being used; or the group name of a suspended group job. The field is used only when the activity field is *PGM, *MNU, *CMD, *S36PRC, *BLDIDX, or *GRP. This field is blank for other activities.

| **Disconnect job allowed indicator**. An indicator as to whether the job is allowed to be disconnected or not using the Disconnect Job (DSCJOB) command. The valid values that will be returned are:

| **1** The job is allowed to be disconnected.
| **0** The job is not allowed to be disconnected.

| **Display station description**. The text that describes the device description of the display station to which the user is signed on.

| **Display station name**. The name of the display station to which the user is currently signed on. This field is used as part of the job name by the system.

| **Display station specified**. The display station or special value specified in the call to this API.

| **Format name**. The name of the format that determines the content of the information returned for each signed-on user. This is specified on the call to this API.

| **Include disconnected jobs and suspended group jobs indicator specified**. The include disconnected jobs and suspended group jobs indicator specified in the call to this API.

| **Include signed-off users with output waiting to print indicator specified**. The include signed-off users with output waiting to print indicator specified in the call to this API.

| **Job number**. The system-assigned job number.

| **Reserved**. An ignored field.

| **User description**. The text of the user profile description for the user currently signed on the system.

| **User name**. The user profile name of the user currently signed on the system.

| **User name specified**. The user name or special value specified in the call to this API.

| **User space library name**. The library in which the user space is located.

| **User space name**. The name of the user space that is to receive the generated list and the name of the library that contains this user space. This is specified in the call to this API.

| Error Messages

| CPF1EA1 E User name parameter is not valid.
| CPF1EA2 E Display station parameter is not valid.
| CPF1EA3 E Include disconnected jobs parameter is not valid.
| CPF1EA4 E Include signed-off users with output parameter not valid.
| CPF1E99 E Unexpected error occurred.
| CPF24B4 E Severe error while addressing parameter list.
| 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.
| CPF811A E User space &4 in &9 damaged.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9804 E Object &2 in library &3 damaged.
| CPF9838 E User profile storage limit exceeded.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Operational Assistant Attention-Key-Handling (group jobs) (QEZMAIN) API

The Operational Assistant Attention-Key-Handling (QEZMAIN) API creates a group job to display the AS/400 Operational Assistant menu (ASSIST). This avoids the Attention-key-handling program running in the same job as an application which leaves the keyboard unlocked between input and output operations. This API can only be used in an interactive job.

Notes:

1. The various job attributes (such as library list or the System/36 environment) may differ between the user's job and the job in which the ASSIST menu runs. To avoid this situation, you might want to use the QEZAST API.
2. This program is only intended to be used as an Attention-handling program for jobs that do not transfer to other group jobs.

There are no parameters for this API.

Error Messages

| CPF1E15 E Problem occurred while calling Operational Assistant.

Operational Assistant Attention-Key-Handling (nongroup jobs) (QEZAST) API

The Operational Assistant Attention-Key-Handling (QEZAST) API uses the GO ASSIST command to display the AS/400 Operational Assistant menu (ASSIST). If you want your users to access the Operational Assistant menu by selecting an option from your application menus, you can add the control language (CL) statement ***call qezast*** to your application. This API can only be used in an interactive job.

This API can be used in place of the Operational Assistant Attention-Key-Handling (QEZMAIN) API when you do not want the Attention key to bring up the ASSIST menu in a group job.

There are no parameters for this API.

Error Messages

| CPF6ACD E Menu &1 in &2 is wrong version for system.
| CPF6AC7 E Menu &1 in library &2 not displayed.

Save Information (QEZSAVIN) API

The Save Information (QEZSAVIN) API displays the Save Information to Help Resolve a Problem display (option 10 on the Operational Assistant Documentation and Problem Handling menu). On that display, users can type a short description of the problem that they are experiencing with the system or with an application. A problem ID is assigned so that a problem analysis person can later look at this information using the Work with Problem (WRKPRB) command. The following information is collected:

- The entries in the QHST history log for the previous hour.
- Printer output from the following commands is placed on the QEZDEBUG output queue:
  - Work with Active Jobs (WRKACTJOB)
  - Display Messages (DSPMSG—For the work station and user)
  - Display System Operator Messages (DSPMSG QSYSOPR)
  - Display Job Log (DSPJOBLOG—For each group job at your display station)
  - Display Job (DSPJOB—For each group job at your display station)
  - Display PTF (DSPPTF—*ALL to give the PTF level of your system)
- Any service dumps (QPSRVDMP), program dumps (QPGMDMP), and job logs (QPJOBLOG) for this user.

There are no parameters for this API. This API can only be used in an interactive job.

Error Messages

| CPF1E99 E Unexpected error occurred.
| CPF9871 E Error occurred while processing.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Send Message (QEZSNDMG) API

Parameters

Optional Parameter Group 1:

1	Message type	Input	Char(10)
2	Delivery mode	Input	Char(10)
3	Message text	Input	Char(*)
4	Length of message text	Input	Binary(4)
5	List of user profile or display station names	Input	Array of Char(10)
6	Number of user profile or display station names	Input	Binary(4)
7	Message sent indicator	Output	Binary(4)
8	Function requested	Output	Binary(4)
9	Error code	I/O	Char(*)

Optional Parameter Group 2:

10	Show Send a Message display	Input	Char(1)
11	Qualified message queue name	Input	Char(20)
12	Name type indicator	Input	Char(4)

The Send Message (QEZSNDMG) API sends a message to one or more users or display stations and optionally shows the Operational Assistant Send a Message display (Figure 49-1) before sending the message. Parameters can be specified, providing the ability to display initial values (defaults) to the user. Parameters determine:

- The type of message that is sent (informational or inquiry)
- The delivery mode of the message (break or normal)
- The message text
- The users or the display stations who will receive the message
- The message queue to receive the reply to an inquiry message
- Whether or not the Send a Message display is shown
- Whether the list of names contains user profile names or display station names

This API combines the functions of the Send Message (SNDMSG) and Send Break Message (SNDBRKMSG) commands. In addition, it provides the ability to:

- Send inquiry messages to more than one user
- Send break messages to user profiles
- Send break and inquiry messages to all active users
- Send nonbreak and inquiry messages to display stations

This API can be called with or without parameters. If parameters are specified, at least nine parameters are required. If this API is called in a batch job, all 12 parameters must be specified, and the message will be sent without using the Send a Message display.

Refer to Chapter 42, "Network Management APIs" on page 42-1 for information on alert messages. Refer to Chapter 40, "Message Handling APIs" on page 40-1 for information on the message handling APIs.

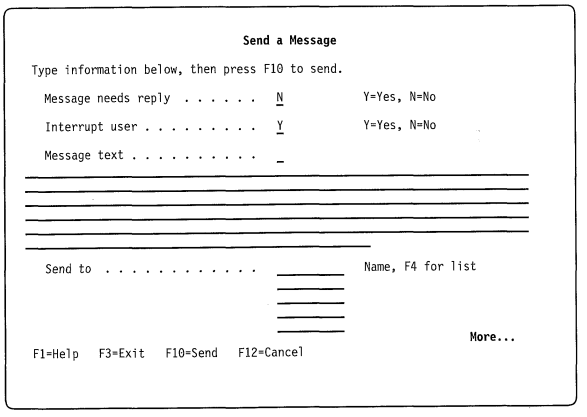


Figure 49-1. Send a Message Display

Optional Parameter Group 1

Message type
INPUT; CHAR(10)
The type of message to send. The value you specify determines the default for the *Message needs reply* prompt on the Send a Message display (N for *INFO and Y for *INQ). You must specify one of these values:

***INFO** Informational. The message does not need a reply.

***INQ** Inquiry. The message needs a reply. If the message queue and library name parameter was specified, the reply is placed on that message queue; otherwise, the reply is placed on the message queue specified in the user profile of the sender.

The user profile name of the person sending the reply is added to the beginning of the message text, allowing the person receiving the reply to determine which user it is from.

Delivery mode
INPUT; CHAR(10)
The delivery mode of the message. The value you specify determines the default for the *Interrupt user* prompt on the Send a Message display (Y for *BREAK and N for *NORMAL). If the user is not authorized to send a break message, *NORMAL is used regardless of the value you specify. You must specify one of these values:

***BREAK** Break message. If the user is signed on, the message goes to the work station message queues that the user is signed on to and temporarily interrupts the work that the user is doing. If the user is not signed on, the message goes to the user profile message queue and the sender is notified. If display station names are specified, the message goes to the message queues for the specified display stations.

***NORMAL** The message goes to the user profile or display station message queue. If the message queue is in notify mode for that user, the message waiting light is turned on. If the message queue is in break mode, the message temporarily interrupts the work that the receiver is doing. If the message queue is in hold mode, the receiver is not notified.

Message text
INPUT; CHAR(*)
The complete text of the message. The text you specify is displayed as the default on the Send a Message display. This must not be blank if used in a batch job or if the Show Send a Message display parameter is N.

Length of message text
INPUT; BINARY(4)
The length of the message text, in bytes. Valid values are 0 through 494. This must be greater than 0 if the API is used in a batch job or if the Send a Message display is not to be shown.

List of user profile or display station names
INPUT; ARRAY OF CHAR(10)
A list of 0 through 299 user profile or display station names to which the message is being sent. The list you specify is shown as the default on the Send a Message display.

The name type indicator parameter indicates whether the names in the list are user profile names or display station names; the default is user profile names. At least one name must be specified if the API is used in a batch job or if the Send a Message display is not to be used.

The message is sent to the user profile or display station message queue. To specify other user message queues, use one of the following special values:

***ALL** The message queues of all users. When you use this value, it must be the only item in the list. This value cannot be used if *DSP is specified for the name type indicator parameter.

***ALLACT** The message queues of all active users or display stations. This value can be used in combination with specific user profile names or display station names and with *SYSOPR.

***SYSOPR** The system operator's message queue, QSYSOPR. This value can be used in combination with specific user profile names and with *ALLACT. It cannot be used if *DSP is specified for the name type indicator parameter.

If the list specifies display station names, the library list will be used to find the work station message queues.

Note: *JOBCTL special authority is required to use the *ALL or *ALLACT value in this parameter.

Number of user profile or display station names
INPUT; BINARY(4)
The number of user profile or display station names specified. Valid values are 0 through 299. When you use the special value *ALL for the list of user profile or display station names parameter, specify 1. This must be greater than 0 if the API is used in a batch job or if the Send a Message display is not to be used.

Message sent indicator
OUTPUT; BINARY(4)
Whether the user pressed F10 (Send message) to send one or more messages from the Send a Message display.

One of the following values is returned:

0 No messages were sent.
1 One or more messages were sent. If the Show Send a Message display parameter is N and the program completes without error, this will always be 1.
2 One or more messages were sent, but one or more of the names specified were not valid. If the Show Send a Message display parameter is Y or not specified, the user names will be displayed and validated before the message is sent; therefore, this value will not be used.

Function requested
OUTPUT; BINARY(4)
The function that the user requested when exiting the Send a Message display.

One of the following values is returned:

-4 User pressed the Exit key (F3).
-8 User pressed the Cancel key (F12).
0 Send a Message display was not 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.

Optional Parameter Group 2

Show Send a Message display
INPUT; CHAR(1)
Whether or not to show the Send a Message display before sending the message.

If this parameter is used, one of the following values must be specified;

Y Use the Send a Message display. This is the default.
N Do not use the Send a Message display. This value must be specified if the program is running in a batch job.

Qualified message queue name
INPUT; CHAR(20)
The name of the message queue that is to receive the reply for inquiry-type messages. The first 10 characters contain the message queue name, and the second 10 characters contain the library name. If this parameter is blank or not specified, the reply will be sent to the message queue specified in the user profile of the sender. This parameter is ignored if the message type parameter is not *INQ.

If this parameter is used, you can use these special values for the library name:

***LIBL** The library list
***CURLIB** The job's current library

Name type indicator
INPUT; CHAR(4)
The type of names in the list

If this parameter is used, one of the following values must be specified:

***USR** The list of names contains only user profile names. *USR is the default.
***DSP** The list of names contains only display station names. This value may not be specified if the Show Send a Message display parameter is Y.

Error Messages

CPF1EA0 E Not authorized to send message to *ALL or *ALLACT.
CPF1EBA E Parameters passed on call do not match those required.
CPF1EBB E Value &1 not valid for list of display station names.
CPF1EBC E At least one user or display station must be specified.
CPF1EBD E Message text cannot be blank.
CPF1EBE E Display station names cannot be used with the Send a Message display.
CPF1EBF E Value for Show Send a Message display not valid.
CPF1EB2 E Delivery mode must be *BREAK or *NORMAL.
CPF1EB3 E Value for length of message text must be 0 through 494.
CPF1EB4 E Number of user profile names not valid.
CPF1EB5 E Value for number of user profile names must be 0 through 299.
CPF1EB6 E Program &1 cannot be run as a batch job.
CPF1EB7 E *ALL not allowed with other user profile names.
CPF1EB8 E Value &1 for name type indicator not valid.
CPF1EB9 E Value &1 not valid for list of user profile names.
CPF1E99 E Unexpected error occurred.
CPF24B3 E Message type &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF2403 E Message queue &1 in &2 not found.
CPF2408 E Not authorized to message queue &1.
CPF3CF1 E Error code parameter not valid.
CPF9871 E Error occurred while processing.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Work with Jobs (QEZBCHJB) API

The Work with Jobs (QEZBCHJB) API uses the Work with User Jobs (WRKUSRJOB JOBTYPE (*BATCH)) command to display either of the following:

- For the basic assistance level, the Work with Jobs display
- For the intermediate assistance level, the Work with User Jobs display

There are no parameters for this API.

Error Messages

Messages are the same as the WRKUSRJOB command messages. For additional information, refer to the *Programming Reference Summary*.

Work with Messages (QEZMSG) API

The Work with Messages (QEZMSG) API uses the Display Messages (DSPMSG) command to display either of the following:

- For the basic assistance level, the Work with Messages display
- For the intermediate assistance level, the Display Messages display

There are no parameters for this API. When the API is called, the default parameter values are used for the command.

Error Messages

Messages are the same as the WRKMSG command messages. For additional information, refer to the *Programming Reference Summary*.

Work with Printer Output (QEZOUTPT) API

The Work with Printer Output (QEZOUTPT) API uses the Work with Spooled Files (WRKSPLF) command to display either of the following:

- For the basic assistance level, the Work with Printer Output display
- For the intermediate assistance level, the Work with All Spooled Files display

There are no parameters for this API. When the API is called, the default parameter values are used for the command.

Error Messages

Messages are the same as the WRKMSG command messages. For additional information, refer to the *Programming Reference Summary*.

Chapter 50. Operational Assistant Exit Programs

This chapter discusses how to tailor some of the Operational Assistant functions to your needs by using exit programs.

The Operational Assistant exit programs consist of:

Tailoring automatic cleanup (QEZUSRCLNP) for running your own cleanup programs (IBM supplied).
Tailoring Operational Assistant backup for running your own backup functions (user supplied).
Tailoring power off (QEZPWROFFP) for changing how you want the system automatically powered on and off (IBM supplied).

Exit Program for Tailoring Automatic Cleanup

You may want to develop your own programs to regularly clean up some of the objects that are not handled by the Operational Assistant automatic cleanup functions.

You can incorporate your own programs into the IBM-supplied automatic cleanup function by using the QEZUSRCLNP program. Then, whenever the system runs automatic cleanup, it also runs your own cleanup programs.

To make a copy of the QEZUSRCLNP program:

1. Type `RTVCLSRC` (the Retrieve CL Source command) on any command line and press F4 (Prompt). Type the following values for the prompts:

Program	`QEZUSRCLNP`
Library	`QSYS`
Source file	`Name of source file`
Library	`Name of library`

   Press the Enter key.
2. Insert statements that run your own cleanup programs into your copy of QEZUSRCLNP.
3. Compile your copy of the QEZUSRCLNP program and store it in a library that appears before the QSYS library in the system part of the library list as specified in system value QSYSLIBL.

Whenever the system runs the automatic cleanup function, your version of QEZUSRCLP is also run.

Exit Program for Tailoring Operational Assistant Backup

Parameters

Required Parameter Group:

1	Calling product	Input	Char(10)
2	Exit indicator	Input	Char(10)
3	Options	Input	Char(10)
4	Device	Input	Char(40)
5	Tape set	Input	Char(4)
6	Return code	Input	Char(7)

You can tailor the Operational Assistant automatic backup by specifying an exit program on the Change Backup (CHGBCKUP) command. If an exit program is specified, that program is called both before the Operational Assistant backup is run and after backup is run. You specify within the exit program when you want your functions to run.

If this program ends abnormally or sends an escape message to its caller when running before the backup, the backup will not continue.

When running backup, you could write your own exit program to back up some additional objects that are not included on the folder or library backup list.

Another example of using an exit program could be when you want to clean up some items before the system does its backup. This would save time and resources by not backing up objects you want deleted.

Refer to "Using the Operational Assistant Exit Program for Operational Assistant Backup" on page A-52 for an example exit program for backup.

Required Parameter Group

Parameters your exit program should be able to handle are:

Calling product
INPUT; CHAR(10)
The name of the product calling the exit program. This parameter is supplied so that the exit program can tell whether it is called from the AS/400 Run Backup (RUNBCKUP) command or from another application. When called from the RUNBCKUP command, the value is QEZBACKUP.

Exit indicator
INPUT; Char(10)
Whether this program is called before or after the backup is done.

The possible values are:

***BEFORE** This call is before the backup has started.
***AFTER** This call is after the backup has run.

Options
INPUT; Char(10)
Indicates that the specified backup options are used.

The possible values are:

***DAILY**	The daily backup options are used.
***WEEKLY**	The weekly backup options are used.
***MONTHLY**	The monthly backup options are used.

Device
Input; Char(40)
The name of up to four devices to be used for the backup. Each device is left-justified on a 10-byte boundary.

Tape set
Input; Char(4)
The name of the tape set to be used for the backup. Operational Assistant backup combines the tape set name (1 to 4 characters) with volume numbers from 01 to 99 to generate the volume IDs of the tape volumes to be used by the backup.

Return code
Input; Char(7)
The message ID of the message returned by backup. This is blank before the backup.

Error Messages

CPC1E62 C Backup successfully completed.
CPF1E68 E Backup incomplete.
CPF1EE7 E Unexpected error occurred during backup.
CPF1E99 E Unexpected error occurred.

Exit Program for Tailoring Power Off

The Power-Off exit program (QEZPWROFFP) is shipped with the system and stored in the QSYS library. This exit program powers off the system according to the power on and off schedule by running the Power Down System command PWRDWNSYS OPTION(*IMMED). (Use the Change Power Schedule Entry (CHGPWRSCDE) command or the Change Power Schedule (CHGPWRSCD) command to set the power on and off schedule.)

You can tailor this program to change how you want the system powered off. For example, you could change the program so that the system would not power off immediately.

To change this program:

1. Type `RTVCLSRC` (the Retrieve CL Source command) on any command line and press F4 (Prompt). Type the following values for the prompts:

Program	`QEZPWROFFP`
Library	`QSYS`
Source file	`Name of source file`
Library	`Name of library`

   Press the Enter key.
2. Change the source to tailor the program. For example, you may want to specify RESTART(*YES) for the PWRDWNSYS command to cause the system to power on (IPL) when the power down is complete.
3. When you are finished changing the program, use the Create CL Program (CRTCLPGM) command and fill in the fields as you did when you retrieved the program. The program you create must be named QEZPWROFFP.
4. To use the new program, put the new program in a library before the QSYS library in the system part of the library list as specified in system value QSYSLIBL.

Notes:

1. Remember that this program controls the scheduled powering off of the system. If you remove the Power Down System (PWRDWNSYS) command, the system does not automatically power down on schedule. If you change the OPTION parameter on the PWRDWNSYS command to *CNTRLD, the system may wait the amount of time specified by the DELAY parameter before it powers off. If the DELAY parameter is changed to *NOLIMIT, the system may never power off.
2. If your next scheduled power-on time is fairly close to the scheduled power-off time, and you have long running commands or calls to programs ahead of the PWRDWNSYS command, your system may not have powered off by the time it is scheduled to power on again. Before the system is powered down, check the time of the next scheduled power up using the Retrieve Power Schedule Entry (RTVPWRSCDE) command. You should also use the Change Power Schedule Entry (CHGPWRSCDE) command to reset the system value. If it is within 1/2 hour, do not power off the system; otherwise, your system may not power on again, and you will need to do a manual IPL.
3. This program will not get called when the current time is less than 30 minutes before the next scheduled power-on time.

Part 17. Performance Collector APIs

Chapter 51. Performance Collector APIs

This chapter discusses the AS/400 performance collector APIs and the Performance Monitor exit program. They are:

List Performance Data (QPMLPFRD) returns data from the data collector to the user space specified in the Work with Collector API.

Work with Collector (QPMWKCOL) controls the starting and stopping of collections of information for certain types of resources. This API allows you to change the way data about a certain resource is collected.

Performance Monitor exit program processes the performance data that is collected by the performance monitor as the monitor ends.

Note: The Work with Collector API must be used before using the List Performance Data API.

The APIs in this chapter are presented in alphabetical order and are followed by the exit program.

List Performance Data (QPMLPFRD) API

Parameters

Required Parameter Group:

1	Type of resource	Output	Char(10)
2	Sequence number of collection	Output	Binary(4)
3	Error code	I/O	Char(*)

The List Performance Data (QPMLPFRD) API returns the latest collection of performance data in the user space specified for the resource on the Work with Collector (QPMWKCOL) API. QPMLPFRD only returns data for one type of a resource at a time. The user cannot specify the type of data that is returned by QPMLPFRD. It returns whatever resource data is ready at the time the QPMLPFRD call is made. The type of resource must be tested to determine its type if more than one type is collected. The call to QPMLPFRD returns the type of resource data and the sequence number of the collection. The sequence number is incremented for every second that has passed and can be used to see if an interval collection was missed. For example, if job data is being collected at 15-second intervals and the previous call to QPMLPFRD returned a sequence number of 265, the sequence number for the next collection retrieved should be 280 or else a collection was missed. This API should be called once per interval for each type of resource data being collected.

The data returned by QPMLPFRD is in a raw format and the user needs to perform delta calculations on the data before it can be used (just as in the sequence number example above). Deltas are the difference between the latest data numbers and the previous data numbers. For example, a user requests job data every 15 seconds using the Work with Collector (QPMWKCOL) API. The user then calls QPMLPFRD to copy the latest collection of data into the resource's user space. The transaction count for JOB1 is 256. In the previous collection, the transaction count for JOB1 was 251. By subtracting the previous data from the current data (256 - 251), the number of transactions performed by JOB1 in that 15 seconds is found. Thus, the user must retain the previous interval's data to calculate deltas from. After the deltas are calculated, the current data becomes the previous data in preparation for the next interval. Because QPMLPFRD replaces the data in the resource's user space with the new data, the user must either save the previous interval's data of interest or use QPMWKCOL to change the resource's user space before calling QPMLPFRD again. This also means that after starting a collection, a user must call QPMLPFRD twice before he can do any calculations.

Using deltas, the impact of missed collections is lessened by the delta calculations. For example, if you were collecting data at 15-second intervals and the sequence numbers for your last two collections were 315 and 345, a collection was missed. You can still perform delta calculations using these two collections and get data for the time period you missed. Of course, the data represents an interval of 30 seconds instead of 15. However, you should never calculate deltas for a period of greater than 4 minutes (the longest collection interval). When the time between collections exceeds 4 minutes, there is a risk that counters will wrap twice. Because there is no way to tell if counters wrapped twice, the user would perform delta calculations as normal and end up with inaccurate data.

There are two items to be careful of when calculating deltas. First, the data between the two collections must be matched by item. This is not a problem for pool, disk, or input/output processor (IOP) data because the number of items (pools, disk arms, IOPs) and their order will not change from collection to collection. However, because jobs can start and end and communications lines can vary on and off during a collection, the number of list items for job and communications data can change from collection to collection. The items are kept in relatively the same order, but when a job ends or a line varies off, it will be removed from the list causing all items after it to move up one position. New jobs and lines that vary on are always added to the end of the lists.

The second situation is when the raw-data counters wrap. For example, assume counter A can hold up to 99 and currently it is set at 96. If the system adds 10 to the count, the value of counter A becomes 6 because when it reaches 100 it starts over again at 0. When a counter wraps, it makes the delta calculation result in a negative number. To compensate for this, a wrap factor equal to the largest number the field can hold should be added to the negative delta. The

following excerpt of code (written in C) shows how a subroutine can be defined to calculate deltas.

```
int CalculateDelta(int CurrentData, int PrevData)
{
  #define MAXBIN4 2147483647
  int Delta;

  Delta = CurrentData - PrevData;
  if (Delta < 0)
    Delta = Delta + MAXBIN4;
  return Delta;
}
```

QPMLPFRD relies on the Work with Collector (QPMWKCOL) API. First, QPMWKCOL must be used to start a collection before QPMLPFRD can be called. Second, the user space specified for a resource on the QPMWKCOL call is the user space that QPMLPFRD will copy the data into.

The data in the user spaces is replaced only if QPMLPFRD is called. However, internal data spaces get updated with every collection. It is these internal data spaces that are copied to the user spaces when QPMLPFRD is called. Therefore, if you do not call QPMLPFRD for every collection every interval, data will be missed. Although, as mentioned above, deltas can help compensate for missed collections, it is not recommended to make a regular practice of skipping collections.

The QPMLPFRD and QPMWKCOL APIs allow multiple users to be able to share the same data collection. This sharing minimizes the system overhead when multiple users are collecting data and ensures that each user is getting data consistent and synchronized with other users.

Authorities and Locks

User Space Authority *USE
Library Authority *USE
User Space Lock *EXCL

Required Parameter Group

Type of resource
OUTPUT; CHAR(10)
The type of resource the collected data is for. It will be set to one of the following values:

***JOB**	Job-related information ("Job Format" on page 51-3)
***POOL**	Pool-related information ("Pool Format" on page 51-5)
***DISK**	Disk-related information ("Disk Format" on page 51-6)
***IOP**	IOP-related information ("IOP Format" on page 51-8)
***COMM**	Communications-related information ("Communications Data Formats" on page 51-9)

Sequence number of collection
OUTPUT; BINARY(4)
The sequence number of this collection of data. This value increases by one for every second.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated List

The list of performance data that the QPMLPFRD API returns into the user space consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

The user area and generic header are described in "User Space Format for List APIs" on page 2-7. For details about the remaining items, see the following sections.

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."

Except where noted, delta calculations need to be performed on all numeric (BINARY(4)) fields.

Input Parameter Section: For a description of the fields in this format, see "Field Descriptions" on page 51-3.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Type of resource
10	A	CHAR(2)	Reserved
12	C	BINARY(4)	Sequence number of collection

Header Section: For a description of the fields in this format, see "Field Descriptions" on page 51-3.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Type of resource
10	A	CHAR(2)	Reserved
12	C	BINARY(4)	Interval time
16	10	BINARY(4)	Total CPU seconds used
20	14	BINARY(4)	Number of CPUs

Offset		Type	Field
Dec	Hex		
24	18	BINARY(4)	Sequence number of collection
28	1C	CHAR(10)	System name
38	26	CHAR(6)	Release level
44	2C	CHAR(12)	Date and time of collection

Field Descriptions

Date and time of collection. The date and time the data collection interval ended. This will be in the format YYMMDDHHMMSS, where:

YY Year
MM Month
DD Day
HH Hour
MM Minute
SS Second

Interval time. The number of seconds in this interval collection.

Number of CPUs. The number of CPUs configured on the system. No delta calculation should be performed on this field.

Release level. The version, release, and modification level of the operating system the data was collected on.

Reserved. An ignored field.

Sequence number of collection. The sequence number of the data collection. This number increases by 1 for every second that has passed.

System name. The name of the system the data was collected on.

Total CPU seconds used. The total number of CPU seconds used during the collection interval.

Type of resource. The type of resource the collected data is for. The possible values for this field are:

**JOB* Job-related information
**POOL* Pool-related information
**DISK* Disk-related information
**IOP* IOP-related information
**COMM* Communications-related information

Job Format: For a description of the fields in this format, see "Job Field Descriptions" on page 51-4.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job name
10	A	CHAR(10)	User name
20	14	CHAR(6)	Job number
26	1A	CHAR(1)	Job type
27	1B	CHAR(1)	Job subtype
28	1C	CHAR(1)	Pass-through source job flag
29	1D	CHAR(1)	Pass-through target job flag
30	1E	CHAR(1)	Emulation job flag
31	1F	CHAR(1)	PC Support application job flag
32	20	CHAR(1)	Target DDM job flag
33	21	CHAR(1)	MRT job flag
34	22	CHAR(1)	System/36 environment job flag
35	23	CHAR(2)	Job priority
37	25	CHAR(2)	Job pool
39	27	CHAR(13)	Reserved
52	34	BINARY(4)	Time slice
56	38	BINARY(4)	CPU time
60	3C	BINARY(4)	Transaction count
64	40	BINARY(4)	Transaction time
68	44	BINARY(4)	Synchronous database reads
72	48	BINARY(4)	Synchronous database writes
76	4C	BINARY(4)	Synchronous nondatabase reads
80	50	BINARY(4)	Synchronous nondatabase writes
84	54	BINARY(4)	Asynchronous database reads
88	58	BINARY(4)	Asynchronous database writes
92	5C	BINARY(4)	Asynchronous nondatabase reads
96	60	BINARY(4)	Asynchronous nondatabase writes
100	64	BINARY(4)	Communications puts
104	68	BINARY(4)	Communications gets
108	6C	BINARY(4)	EAO exceptions
112	70	BINARY(4)	Binary overflows
116	74	BINARY(4)	Decimal overflows
120	78	BINARY(4)	Floating-point overflows
124	7C	BINARY(4)	Logical database reads
128	80	BINARY(4)	Logical database writes
132	84	BINARY(4)	Miscellaneous database operations

Offset			
Dec	Hex	Type	Field
136	88	BINARY(4)	Permanent writes
140	8C	BINARY(4)	Checksum I/O
144	90	BINARY(4)	PAG faults
148	94	BINARY(4)	Number of print lines
152	98	BINARY(4)	Number of print pages
156	9C	BINARY(4)	Active-to-wait transitions
160	A0	BINARY(4)	Wait-to-ineligible transitions
164	A4	BINARY(4)	Active-to-ineligible transitions
168	A8	CHAR(10)	Line description
178	B2	CHAR(10)	Secondary line description

Job Field Descriptions

Active-to-ineligible transitions. The total number of transitions from active state to ineligible state.

Active-to-wait transitions. The total number of transitions from active state to wait state.

Asynchronous database reads. The total number of physical asynchronous read operations for database functions.

Asynchronous database writes. The total number of physical asynchronous write operations for database functions.

Asynchronous nondatabase reads. The total number of physical asynchronous read operations for nondatabase functions.

Asynchronous nondatabase writes. The total number of physical asynchronous write operations for nondatabase functions.

Binary overflows. The number of binary overflows.

Checksum I/O. The total number of I/Os (read and write operations) performed for checksum updating due to protected write operations caused by this job.

CPU time. The processing unit time (in milliseconds) used by this job.

Communications gets. The number of communications read (logical) operations. These do not include remote work station activity. They include only activity related to intersystem communication function (ICF) files when the I/O is for an ICF device.

Communications puts. The number of communications writes. These do not include remote work station activity. They include only activity related to ICF files when the I/O is for an ICF device.

Decimal overflows. The number of decimal overflows.

EAO exceptions. The number of effective address overflow exceptions.

Emulation job flag. This field will be set to 1 if this is a emulation job. Otherwise, it will be set to 0.

Floating-point overflows. The number of floating point overflows.

Job name. The name of the job as identified to the system. For an interactive job, the system assigns the job the name of the work station where the job started. For a batch job, the name is specified in the command when the job is submitted.

Job number. The number of the job. This number is assigned by the system only for jobs. Tasks do not have a job number.

Job pool. The pool that the job ran in.

Job priority. The job's priority over other jobs.

Job subtype. The subtype of the job. The possible values for this file are:

blank	The job has no special subtype
D	Immediate
E	Evoke job (communications batch)
J	Prestart job
P	Print driver job
T	Multiple requester terminal (MRT) job (System/36 environment only)
U	Alternate spool user

Job type. The type of job or task. The possible values for this field are:

A	Autostart job
B	Batch job
H	Horizontal Licensed Internal Code (HLIC) (tasks only)
I	Interactive job
M	Subsystem monitor job
R	Spooled reader job
S	System job
V	Vertical Licensed Internal Code (VLIC) (tasks only)
W	Spooled writer job
X	Start-control-program-function (SCPF) system job

Line description. The name of the communications line this work station and its controller are attached to. This is only available for remote work stations.

Logical database reads. The number of times the database module was called. This does not include I/O operations to readers/writers, or I/O operations caused by the Copy Spooled File (CPYSPLF) or Display Spooled File (DSPSPLF) command. If SEQONLY(*YES) is specified on the Override with Database File (OVRDBF) command, these numbers show each block of records read, not the number of individual numbers read.

Logical database writes. The number of times the internal database write function was called. This does not include I/O operations to readers/writers, or I/O operations caused by the CPYSPLF or DSPSPLF commands. If SEQONLY(*YES) is specified on OVRDBF command, these numbers show each block of records written, not the number of individual records written.

MRT job flag. This field will be set to 1 if this is a multiple requester terminal (MRT) job. Otherwise, it will be set to 0.

Miscellaneous database operations. The number of update, delete, force-end-of-data, and release operations.

Number of print lines. The number of lines written by the program. This does not reflect what is actually printed. Spooled files can be ended or printed with multiple copies.

Number of print pages. The number of pages printed by the program.

PAG faults. The total number of times the process access group (PAG) was referred to, but was not in main storage.

Permanent writes. The number of permanent writes.

PC Support application job flag. This field will be set to 1 if this is a PC Support application job. Otherwise, it will be set to 0.

Pass-through source job flag. This field will be set to 1 if this is a pass-through source job. Otherwise, it will be set to 0.

Pass-through target job flag. This field will be set to 1 if this is a pass-through target job. Otherwise, it will be set to 0.

Reserved. An ignored field.

Secondary line description. The name of the communications line this work station and its controller are attached to. This is only available for pass-through and emulation.

Synchronous database reads. The total number of physical synchronous read operations for database functions.

Synchronous database writes. The total number of physical synchronous write operations for database functions.

Synchronous nondatabase reads. The total number of physical synchronous read operations for nondatabase functions.

Synchronous nondatabase writes. The total number of physical synchronous write operations for nondatabase functions.

System/36 environment job flag. This field will be set to 1 if this is a System/36 environment job. Otherwise, it will be set to 0.

Target DDM job flag. This field will be set to 1 if this is a target DDM job. Otherwise, it will be set to 0.

Time slice. The time slice value in seconds. No delta calculation should be performed on this field.

Transaction count. The number of transactions performed by the job.

Transaction time. The total transaction time (in seconds).

User name. The user profile under which the job is run. The user name is the same as the user profile name and can come from several different sources depending on the type of job.

Wait-to-ineligible transitions. Total number of transitions from wait state to ineligible state.

Note: Currently no data is collected for a job in the interval it ends.

Pool Format: For a description of the fields in this format, see "Pool Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Pool number
4	4	BINARY(4)	Activity level
8	8	BINARY(4)	Pool size
12	C	BINARY(4)	Machine-reserved portion
16	10	BINARY(4)	Database faults
20	14	BINARY(4)	Nondatabase faults
24	18	BINARY(4)	Database pages
28	1C	BINARY(4)	Nondatabase pages
32	20	BINARY(4)	Active-to-wait transitions
36	24	BINARY(4)	Wait-to-ineligible transitions
40	28	BINARY(4)	Active-to-ineligible transitions

Pool Field Descriptions

Active-to-ineligible transitions. The total number of transitions by processes assigned to this pool from active state to ineligible state.

Active-to-wait transitions. The total number of transitions by processes assigned to this pool from active state to wait state.

Activity level. The maximum number of processes that can be active in the machine at the same time. No delta calculation should be performed on this field.

Database faults. The total number of interruptions to processes (not necessarily assigned to this pool) that were required to transfer data into the pool to permit the MI instruction to process the database function.

Database pages. The total number of pages of database data transferred from auxiliary storage to the pool to permit the instruction to run as a consequence of set access state, implicit access group movement, and internal machine actions.

Machine-reserved portion. The amount of storage (in kilobytes) from the pool that is dedicated to machine functions. No delta calculation should be performed on this field.

Nondatabase faults. The total number of interruptions to processes (not necessarily assigned to this pool) that were required to transfer data into the pool to permit the MI instruction to process the nondatabase function.

Nondatabase pages. The total number of pages of nondatabase data transferred from auxiliary storage to the pool to permit the instruction to run as: a consequence of set access state, implicit access group movement, and internal machine actions.

Pool number. The unique identifier of this pool. The value is from 1 to 16. No delta calculation should be performed on this field.

Pool size. The amount of main storage (in kilobytes) assigned to the pool. No delta calculation should be performed on this field.

Wait-to-ineligible transitions. Total number of transitions by processes assigned to this pool from wait state to ineligible state.

Disk Format: For a description of the fields in this format, see "Disk Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(4)	Disk arm number
12	C	CHAR(4)	Disk drive type
16	10	CHAR(1)	Mirror flag
17	11	CHAR(1)	Mirror status
18	12	CHAR(2)	Reserved
20	14	BINARY(4)	Times the arm not busy
24	18	BINARY(4)	Samples taken
28	1C	BINARY(4)	Drive capacity
32	20	BINARY(4)	Drive available space
36	24	BINARY(4)	Blocks read
40	28	BINARY(4)	Blocks written
44	2C	BINARY(4)	Read commands
48	30	BINARY(4)	Write commands
52	34	BINARY(4)	Processor idle loop count
56	38	BINARY(4)	Processor idle loop time
60	3C	BINARY(4)	Seeks > 2/3 of disk
64	40	BINARY(4)	Seeks > 1/3 and < 2/3 of disk
68	44	BINARY(4)	Seeks > 1/6 and < 1/3 of disk
72	48	BINARY(4)	Seeks > 1/12 and < 1/6 of disk
76	4C	BINARY(4)	Seeks < 1/12 of disk
80	50	BINARY(4)	Zero seeks
84	54	BINARY(4)	Buffer overruns
88	58	BINARY(4)	Buffer underruns
92	5C	BINARY(4)	Average queue length
96	60	CHAR(2)	ASP number
98	62	CHAR(2)	Checksum number
100	64	BINARY(4)	Permanent space capacity
104	68	BINARY(4)	Permanent space available

Disk Field Descriptions

ASP number. The auxiliary storage pool (ASP) to which this unit is currently allocated. The values are:

0	This unit is currently not allocated.
1	The system ASP.
2–16	A user ASP.

Average queue length. The number of I/O operations waiting service at sample time. This number includes the I/O operation that is in progress. Divide this by the number of samples taken to get the average queue length.

Blocks read. The number of blocks read. The block length is 520 bytes, which includes 8 bytes of system control information.

Blocks written. The number of blocks written. The block length is 520 bytes, which includes 8 bytes of system control information.

Buffer overruns. The number of times that data was available to be read into the disk controller buffer from the disk, but the disk controller buffer still contained valid data that was not retrieved by the storage device controller. Consequently, the disk had to take an additional revolution until the buffer was available to accept data.

Buffer underruns. The number of times that the disk controller was ready to transfer data to the disk on a write operation, but the disk controller buffer was empty. The data was not transferred in time by the disk IOP to the disk controller buffer. The disk was forced to take an extra revolution awaiting the data.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be done on this field.

Checksum number. The checksum set to which this unit is currently allocated. The values are:

0 The unit is currently not assigned to a checksum set.
1–16 A checksum set.

Disk arm number. The unique identifier of the unit. Each actuator arm on the disk drives available to the machine represents a unit of auxiliary storage. The value of the unit number is assigned by the system when the unit is allocated to an ASP.

Disk drive type. The type of disk drive, such as 9332, 9335, or 6100.

Drive available space. The total number of kilobytes of auxiliary storage space that is not currently assigned to objects or to internal machine functions. Therefore, the space is available on the unit if the ASP containing it is not under checksum protection. No delta calculations should be done on this field.

Drive capacity. The total number of kilobytes of auxiliary storage provided on the unit for the storage of objects and internal machine functions when the ASP containing it is not under checksum protection. The unit reserved system space value is subtracted from the unit capacity to calculate this capacity. No delta calculations should be done on this field. When the ASP containing the unit is under checksum protection, the unit checksum information describes how much of the unit capacity is used for:

- Protected space
- Unprotected space
- Redundant data

IOP address. The IOP bus address. This number can range from 0 through 31. No delta calculation should be done on this field.

Mirror flag. The flag indicating whether this disk arm is mirrored. The values are:

blank The arm is not mirrored.
A The first arm of a mirrored pair.
B The second arm of a mirrored pair.

Mirror status. The status of file mirroring. The values are:

1 Active
2 Resuming
3 Suspended

Permanent space available. The number of kilobytes of permanent space on auxiliary storage available for allocation on the unit that is not currently assigned to objects of internal machine functions. This field has a value other than zero only if this unit is allocated to a checksum set. Units not allocated to a checksum set contain no protected storage area. No delta calculations should be done on this field.

Permanent space capacity. The number of kilobytes of auxiliary storage formatted for storage of protected data on the unit. This field has a value other than zero if this unit is allocated to a checksum set. Units not allocated to a checksum set contain no permanent storage area. This value does not include the size of any data redundancy area, which may also be formatted on the unit. No delta calculations should be done on this field.

Processor idle loop count. The number of times the disk controller passed through the idle loop. The count is increased differently for the 9332 and 9335 disk drives. For the 9332, this counter is increased only if the disk controller is totally idle (no I/O operations are active). For the 9335, even though the disk controller may be idle and the counter gets increased, an I/O operation can be active (for example, seek is being performed). This field is 0 for Disk Unit type 6100.

Processor idle loop time. The time (in hundredths of microseconds) to make one pass through the idle loop. This field is 0 for Disk Unit type 6100. No delta calculation should be done on this field.

Read commands. The number of read data commands.

Reserved. An ignored field.

Samples taken. The number of samples taken at approximately two per second.

Seeks < 1/12 of disk. The number of times the arm traveled from its current position to less than 1/12 of the disk on a seek request.

Seeks > 1/12 and < 1/6 of disk. The number of times the arm traveled more than 1/12 but less than 1/6 of the disk on a seek request.

Seeks > 1/6 and < 1/3 of disk. The number of times the arm traveled more than 1/6 but less than 1/3 of the disk on a seek request.

Seeks > 1/3 and < 2/3 of disk. The number of times the arm traveled more than 1/3 but less than 2/3 of the disk on a seek request.

Seeks > 2/3 of disk. The number of times the arm traveled more than 2/3 of the disk on a seek request.

Times the arm not busy. The number of times there were no outstanding I/O operations active at sample time.

Write commands. The number of write data commands.

Zero seeks. The number of times the access arm did not physically move on a seek request. The operation may have resulted in a head switch.

IOP Format: For a description of the fields in this format, see "IOP Field Descriptions" on page 51-8.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(1)	IOP type
9	9	CHAR(4)	Resource type
13	D	CHAR(3)	Reserved
16	10	BINARY(4)	Idle loop count
20	14	BINARY(4)	Idle loop time
24	18	BINARY(4)	RAM utilization[1,2]
28	1C	BINARY(4)	IOP system function time[1,2]
32	20	BINARY(4)	All protocols communications time[1,2]
36	24	BINARY(4)	SDLC communications time[1,2]
40	28	BINARY(4)	Asynchronous communications time[1,2]
44	2C	BINARY(4)	Bisynchronous communications time[1,2]
48	30	BINARY(4)	X.25 LLC communications time[1,2]
52	34	BINARY(4)	X.25 PLC communications time[1,2]
56	38	BINARY(4)	X.25 DLC communications time[1,2]
60	3C	BINARY(4)	LAN communications time[1,2]
64	40	BINARY(4)	SDLC short-hold mode[1,2]
68	44	BINARY(4)	ISDN LAPE and LAPD time[1,2]
72	48	BINARY(4)	ISDN Q931 communications time[1,2]
76	4C	BINARY(4)	Disk time[2]

Notes:

1 Communication input/output processor (CIOP)
2 Multifunction input/output processor (MIOP)

IOP Field Descriptions

All protocols communications time. The total processing unit time (in milliseconds) used by all the communications protocols. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Asynchronous communications time. The total processing unit time (in milliseconds) used by asynchronous communications tasks. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Bisynchronous communications time. The total processing unit time (in milliseconds) used by bisynchronous communications tasks. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be done on this field.

Disk time. The total processing unit time (in milliseconds) used by disk tasks. This field only applies to multifunction IOPs. Otherwise, it will be set to 0.

Idle loop count. The number of times the IOP ran an idle loop. This is done when the IOP has no work to perform. This count is used with idle loop time.

Idle loop time. The time (in hundredths of microseconds) to run the idle loop once. No delta calculation should be done on this field.

IOP address. The IOP bus address. This number can range from 0 through 31. No delta calculation should be done on this field.

IOP system function time. The total time (in milliseconds) used by the IOP for basic system function. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

IOP type. The type of IOP. The possible values for this field are:

C Communications IOP
D Disk IOP
L Local work station IOP
M Multifunction IOP

ISDN LAPE and LAPD time. The total processing unit time (in milliseconds) used by integrated services digital network (ISDN) communications tasks. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0. The ISDN communications tasks are:

LAPD Link access procedure D-channel
LAPE Enhanced version of LAPD

ISDN Q.931 communications time. The total processing unit time (in milliseconds) used by ISDN Q.931 communications tasks. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

LAN communications time. The total processing unit time (in milliseconds) used by token-ring network and Ethernet communications tasks. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

RAM utilization. Available local storage (in bytes). The number of bytes of free local storage in the IOP. The free local storage will probably be noncontiguous because of fragmentation. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0. No delta calculations should be done on this field.

Reserved. An ignored field.

Resource type. The model number of the IOP.

SDLC communications time. The total processing unit time (in milliseconds) used by SDLC communications tasks. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

SDLC short-hold mode. In SNA, a mode specified during configuration that allows the DTE to connect or reconnect when no data is being transmitted over an X.21 circuit-switched line, while maintaining the logical connection of the SNA sessions across the circuit.

X.25 DLC communications time. The total processing unit time (in milliseconds) used by X.25 data link control (DLC) communications tasks. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

X.25 LLC communications time. The total processing unit time (in milliseconds) used by X.25 logical link control (LLC) communications tasks. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

X.25 PLC communications time. The total processing unit time (in milliseconds) used by X.25 packet layer communications (PLC) tasks. This field only applies to communications and multifunction IOPs. Otherwise, it will be set to 0.

Communications Data Formats: The formats for communications data are handled differently from the formats for other types of resources. All communications protocols are kept in the same space, but, because each protocol has unique data fields, each individual field in the space will have a different meaning depending on the protocol. Therefore, different formats are presented for each protocol. Also, because the protocols vary in the number of data fields, some protocol formats will be padded with blanks at the end so that each record in the space has the same length. The communications data formats are:

- Asynchronous Format ("Asynchronous Format")
- Bisynchronous Format ("Bisynchronous Format" on page 51-10)
- Establishment Communications Link (ECL) Format ("Establishment Communications Link (ECL) Format" on page 51-11)
- Ethernet Format ("Ethernet Format" on page 51-13)
- IDLC Format ("IDLC Format" on page 51-14)
- LAPD Format ("LAPD Format" on page 51-15)
- SDLC Format ("SDLC Format" on page 51-17)
- X.25 Format ("X.25 Format" on page 51-18)

Asynchronous Format: For a description of the fields in this format, see "Asynchronous Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	Protocol data units transmitted
56	38	BINARY(4)	Protocol data units received
60	3C	BINARY(4)	Protocol data units received in error
64	40	CHAR(132)	Reserved

Asynchronous Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be performed on this field.

Bytes received. The number of bytes received (data and control characters), including characters received in error.

Bytes transmitted. The number of bytes transmitted (data and control characters) including bytes transmitted again because of errors.

IOP address. The IOP bus address. This number can range from 0 through 31. No delta calculation should be performed on this field.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Number of vary on operations. The number of vary on operations the line had in the collection interval.

Protocol. Protocol type. This will be set to A for asynchronous.

Protocol data units received. The total number of protocol data units received.

Protocol data units received in error. The total number of protocol data units received with parity and stop bit errors.

Protocol data units transmitted. The total number of protocol data units successfully transmitted and the data circuit-terminating equipment (DCE) acknowledged.

Bisynchronous Format: For a description of the fields in this format, see "Bisynchronous Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	Data characters transmitted
56	38	BINARY(4)	Data characters received
60	3C	BINARY(4)	Data characters retransmitted
64	40	BINARY(4)	Data characters received in error
68	44	BINARY(4)	Characters received in error
72	48	BINARY(4)	NAK received to text sent
76	4C	BINARY(4)	Wrong ACK to text sent
80	50	BINARY(4)	Enqueue to text sent
84	54	BINARY(4)	Invalid (unrecognized) format
88	58	BINARY(4)	Enqueue to ACK
92	5C	BINARY(4)	Disconnect received (abort)
96	60	BINARY(4)	EOT received (abort)
100	64	BINARY(4)	Disconnect received (forward abort)
104	68	BINARY(4)	EOT received (forward abort)
108	6C	BINARY(4)	Data blocks transmitted
112	70	BINARY(4)	Data blocks received
116	74	CHAR(80)	Reserved

Bisynchronous Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be done on this field.

Bytes received. The number of bytes received (data and control characters), including bytes received in error.

Bytes transmitted. The number of bytes transmitted (data and control characters) including bytes transmitted again because of errors.

Characters received in error. The number of characters received with a block-check character error.

Data blocks received. The number of data blocks received.

Data blocks transmitted. The number of data blocks transmitted.

Data characters received. The number of data characters received successfully (excluding synchronous characters) while in data mode.

Data characters received in error. The number of data characters received with a block-check character error while in data mode.

Data characters retransmitted. The number of data characters transmitted again.

Data characters transmitted. The number of data characters transmitted successfully while in data mode.

Disconnect received (abort). The number of times the remote station issued a disconnect with abnormal end. This could occur when error recovery did not succeed or the binary synchronous job was ended.

Disconnect received (forward abort). The number of times the host station issued a disconnect with abnormal end. This could occur when error recovery did not succeed or the binary synchronous job was ended.

EOT received (abort). The end of transmission was received (abnormal end). This is similar to a disconnect.

EOT received (forward abort). The end of transmission was received (forward abnormal end). This is similar to a disconnect.

Enqueue to ACK. Enqueue to acknowledged character. The remote station returned an acknowledgment (for example, ACK0), and the host system sent an ENQ character. This indicates that the host station did not recognize the acknowledgment as a valid acknowledgment.

Enqueue to text sent. The number of times text was sent by a station and an ENQ character was returned. The receiving station expected some form of acknowledgment, such as an ACK0, ACK1, or NAK.

IOP address. The IOP bus address. This number can range from 0 through 31. No delta calculation should be done on this field.

Invalid (unrecognized) format. The number of times one of the delimiter characters that encloses the data in brackets being sent or received is not valid.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

NAK received to text sent. Negative acknowledgment character received to text sent. The number of times the remote station did not understand the command sent from the host system.

Number of vary on operations. The number of vary on operations the line had in the collection interval.

Protocol. The protocol type. This will be set to B for bisynchronous.

Reserved. An ignored field.

Wrong ACK to text sent. Wrong acknowledgment character to text sent. The host system received an acknowledgment from the remote device that was not expected. For example, the host system expected an ACK0 and received an ACK1.

Establishment Communications Link (ECL)

Format: For a description of the fields in this format, see "ECL Field Descriptions" on page 51-12.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	I-frame characters transmitted
48	30	BINARY(4)	I-frame characters received
52	34	BINARY(4)	RNR frames transmitted
56	38	BINARY(4)	RNR frames received
60	3C	BINARY(4)	Reject frames transmitted
64	40	BINARY(4)	Reject frames received
68	44	BINARY(4)	I-frames transmitted
72	48	BINARY(4)	I-frames received
76	4C	BINARY(4)	SABME frames transmitted
80	50	BINARY(4)	SABME frames received
84	54	BINARY(4)	N2 retries expiration count
88	58	BINARY(4)	T1 timer expiration count
92	5C	BINARY(4)	Frames transmitted
96	60	BINARY(4)	Frames received
100	64	BINARY(4)	Routing I-frames transmitted
104	68	BINARY(4)	Routing I-frames received
108	6C	BINARY(4)	Line errors
112	70	BINARY(4)	Internal errors
116	74	BINARY(4)	Burst error
120	78	BINARY(4)	ARI/FCI error
124	7C	BINARY(4)	Abort delimiter
128	80	BINARY(4)	Lost frame
132	84	BINARY(4)	Receive congestion
136	88	BINARY(4)	Frame-copied error
140	8C	BINARY(4)	Frequency error
144	90	BINARY(4)	Token error
148	94	BINARY(4)	Direct memory access bus error
152	98	BINARY(4)	Direct memory access parity error
156	9C	BINARY(4)	Address not recognized
160	A0	BINARY(4)	Frame-not-copied error
164	A4	BINARY(4)	Transmit strip error
168	A8	BINARY(4)	Unauthorized AP
172	AC	BINARY(4)	Unauthorized MAC frame
176	B0	BINARY(4)	Soft error
180	B4	BINARY(4)	Transmit beacon
184	B8	BINARY(4)	IOA status overrun
188	BC	BINARY(4)	Frames discarded
192	C0	BINARY(4)	Spurious interrupts

ECL Field Descriptions

ARI/FCI error. Address-recognized indicator or frame-copied indicator error. This is a physical control field-extension field error.

Abort delimiter. The number of times an abnormal ending delimiter was transmitted because of an internal error.

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Address not recognized. Total number of frames with address-not-recognized error.

Burst error. The number of burst errors. Burst of same polarity is detected by the physical unit after the starting delimiter of a frame or token.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be performed on this field.

Direct memory access bus error. Direct memory access (DMA) error for the IOP/IOA bus.

Direct memory access parity error. DMA parity error for the IOP/IOA.

Frame-copied error. The number of times a frame with a specific destination address was copied by another adapter.

Frames discarded. The total number of frames discarded.

Frame-not-copied error. Total number of frames with frame not copied error.

Frames received. The total number of frames (logical link control (LLC) and medium access control (MAC)) received.

Frames transmitted. The total number of frames (LLC and MAC) transmitted.

Frequency error. The number of frequency errors on the adapter.

I-frame characters received. The total number of characters received in all I-frames.

I-frame characters transmitted. The total number of characters transmitted in all I-frames.

I-frames received. The number of I-frames received.

I-frames transmitted. The number of I-frames transmitted excluding I-frames transmitted again.

IOA status overrun. The number of adapter interrupt status queue overruns. The earliest results are discarded.

IOP address. The IOP bus address. This number can range from 0 through 31. No delta calculation should be performed on this field.

Internal errors. The number of adapter internal errors.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line errors. The number of code violations of frame-check sequence errors.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Lost frame. The number of times the adapter could not remove its own frame from the ring.

N2 retries expiration count. This count is updated when the host has attempted to contact a station *n* times, and the T1 timer ended *n* times before the station responded.

Number of vary on operations. The number of vary on operations the line had in the collection interval.

Protocol. Protocol type. This will be set to E for establishment communications link (ECL).

Receive congestion. The number of times a frame was not copied because no buffer was available for the IOA to receive.

Reject frames received. The number of reject frames received.

Reject frames transmitted. The number of reject frames transmitted.

Reserved. An ignored field.

Routing I-frames received. Total number of frames (LLC and MAC) with a routing-information field received.

Routing I-frames transmitted. Total number of frames (LLC and MAC) with a routing-information field transmitted.

RNR frames received. The number of receive-not-ready frames received.

RNR frames transmitted. The number of receive-not-ready frames transmitted.

SABME frames received. The number of set-asynchronous-balanced-mode-extended frames received.

SABME frames transmitted. The number of set-asynchronous-balanced-mode-extended frames transmitted.

Soft error. The total number of soft errors as reported by the adapter. A soft error is an intermittent error on a network that requires retransmission.

Spurious interrupts. The total number of interrupts that medium access control (MAC) could not decode.

T1 timer expiration count. The number of times the T1 timer ended.

Token error. When this adapter serves as ring monitor, the number of times the adapter token timer ended without detecting any frames or tokens on the ring.

Transmit beacon. The total number of beacon frames transmitted.

Transmit strip error. The total number of adapter-frame-transmit or frame-stripping-process errors.

Unauthorized AP. Unauthorized access priority. The number of times the access priority request is not authorized.

Unauthorized MAC frame. The number of unauthorized MAC frames. The adapter is not authorized to send a MAC frame if:

- A source class is specified.
- The MAC frame has a source class of zero.
- MAC frame physical-control-attention field is greater than 1.

Ethernet Format: For a description of the fields in this format, see "Ethernet Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	I-frame characters transmitted
48	30	BINARY(4)	I-frame characters received
52	34	BINARY(4)	RNR frames transmitted
56	38	BINARY(4)	RNR frames received
60	3C	BINARY(4)	Reject frames transmitted
64	40	BINARY(4)	Reject frames received
68	44	BINARY(4)	I-frames transmitted
72	48	BINARY(4)	I-frames received
76	4C	BINARY(4)	SABME frames transmitted
80	50	BINARY(4)	SABME frames received
84	54	BINARY(4)	N2 retries expiration count
88	58	BINARY(4)	T1 timer expiration count
92	5C	BINARY(4)	Total frames transmitted
96	60	BINARY(4)	Total frames received
100	64	BINARY(4)	Inbound frames missed
104	68	BINARY(4)	CRC error
108	6C	BINARY(4)	More than 16 retries
112	70	BINARY(4)	Out-of-window collisions
116	74	BINARY(4)	Alignment error
120	78	BINARY(4)	Carrier loss
124	7C	BINARY(4)	Time domain reflectometry
128	80	BINARY(4)	Receive buffer errors
132	84	BINARY(4)	Spurious interrupts
136	88	BINARY(4)	Discarded inbound frames
140	8C	BINARY(4)	Receive overruns
144	90	BINARY(4)	Memory error
148	94	BINARY(4)	Interrupt overrun
152	98	BINARY(4)	Transmit underflow
156	9C	BINARY(4)	Babble errors
160	A0	BINARY(4)	Signal quality error
164	A4	BINARY(4)	More than one retry to transmit
168	A8	BINARY(4)	Exactly one retry to transmit
172	AC	BINARY(4)	Deferred conditions
176	B0	CHAR(20)	Reserved

Ethernet Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Alignment error. The number of times an inbound frame contained a noninteger number of bytes and a cyclic-redundancy-check (CRC) error.

Babble errors. The number of times the transmitter exceeded the maximum allowable time on the channel.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be performed on this field.

Carrier loss. Access to the network has been disconnected.

| **CRC error**. The number of cyclic-redundancy-check (CRC) errors detected by the receiver.

| **Deferred conditions**. The number of times the chip set on the IOAs deferred transmission due to a busy channel.

| **Discarded inbound frames**. The number of receiver-discarded frames due to the lack of queue entries.

| **Exactly one retry to transmit**. The number of frames that required one retry for successful transmission.

| **I-frame characters received**. The total number of characters received in all I-frames.

| **I-frame characters transmitted**. The total number of characters transmitted in all I-frames.

| **I-frames received**. The number of I-frames received.

| **I-frames transmitted**. The number of I-frames transmitted excluding I-frames transmitted again.

| **IOP address**. The IOP bus address. This number can range from 0 through 31. No delta calculation should be performed on this field.

| **Inbound frames missed**. The number of times a receive buffer error or missed frame was detected by the IOA.

| **Interrupt overrun**. The number of interrupts not processed due to lack of status queue entries.

| **Line active**. The state of the line when the collection interval ended. The values are:

| *0* The line is not active.
| *1* The line is active.

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

| **Line speed**. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

| **Memory error**. The number of times the IOA did not receive a ready signal within 25.6 μ of asserting the address on the data or address lines.

| **More than one retry to transmit**. The number of frames that required more than one retry for successful transmission.

| **More than 16 retries**. The number of frames unsuccessfully transmitted due to excessive retries.

| **N2 retries expiration count**. This count is updated when the host has attempted to contact a station *n* times, and the T1 timer ended *n* times before the station responded.

| **Number of vary on operations**. The number of vary on operations the line had in the collection interval.

| **Out-of-window collisions**. The number of collisions that occurred after the allotted time interval for a collision to occur and after the transmission attempt to be retried.

| **Protocol**. Protocol type. This will be set to T for Ethernet.

| **Reserved**. An ignored field.

| **RNR frames received**. The number of receive-not-ready frames received.

| **RNR frames transmitted**. The number of receive-not-ready frames transmitted.

| **Receive buffer errors**. The number of hardware buffer overflows that occurred upon receiving a frame.

| **Receive overruns**. The number of times the receiver has lost all or part of an incoming frame due to buffer shortage.

| **Reject frames received**. The number of reject frames received.

| **Reject frames transmitted**. The number of reject frames transmitted.

| **SABME frames received**. The number of set-asynchronous-balanced-mode-extended frames received.

| **SABME frames transmitted**. The number of set-asynchronous-balanced-mode-extended frames transmitted.

| **Signal quality error**. The number of times a signal indicating the transmit is successfully complete did not arrive within 2 μ of successful transmission.

| **Spurious interrupts**. The number of times an interrupt was received but could not be decoded into a recognizable interrupt.

| **T1 timer expiration count**. The number of times the T1 timer ended.

| **Time domain reflectometry**. Counter used to approximate distance to a cable fault. This value is associated with the last occurrence of more than 16 retries.

| **Total frames received**. The total number of type II frames received.

| **Total frames transmitted**. The total number of type II frames transmitted.

| **Transmit underflow**. The number of times the transmitter has truncated a message due to the late data received from main storage.

| **IDLC Format:** For a description of the fields in this format, see "IDLC Field Descriptions" on page 51-15.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(10)	Network interface description
29	1D	CHAR(1)	Line active
30	1E	CHAR(2)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	Receive CRC errors
56	38	BINARY(4)	Short frame errors
60	3C	BINARY(4)	Aborts received
64	40	BINARY(4)	Sequence errors
68	44	BINARY(4)	Frames transmitted
72	48	BINARY(4)	Frames retransmitted
76	4C	BINARY(4)	Frames received
80	50	BINARY(4)	Frames received in error
84	54	CHAR(1)	B1 channel
85	55	CHAR(1)	B2 channel
86	56	CHAR(110)	Reserved

IDLC Field Descriptions

Aborts received. The number of frames received that contained high-level data link control (HDLC) abort indicators. This indicates that the remote equipment ended frames before they were complete.

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

B1 channel. The user can send data on this channel. This is set to 1 if the B1 channel was used.

B2 channel. The user can send data on this channel. This is set to 1 if the B2 channel was used.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be performed on this field.

Bytes received. The total number of bytes received from the remote link station. This includes no errors.

Bytes transmitted. The total number of bytes transmitted to a remote link station. This includes bytes retransmitted and bytes sent on transmissions stopped by transmit underrun, in addition to successful transmissions.

Frames received. Total number of information (I), unnumbered information (UI), and supervisory (S) frames received from the remote link station. This includes no errors.

Frames received in error. The sum of receive CRC errors, short frame errors, overrun, underrun, aborts received, and frame sequence errors.

Frames retransmitted. The number of frames that required retransmission due to errors. Errors can be caused by a remote device that is failing or by not receiving data fast enough.

Frames transmitted. Total number of information (I), unnumbered information (UI), and supervisory (S) frames sent to a remote link station. This includes frames retransmitted and frames sent on transmissions stopped by transmit underruns, in addition to successful transmissions.

IOP address. The IOP bus address. This number can range from 0 through 31. No delta calculation should be performed on this field.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Network interface description. The name of the network interface description.

Number of vary on operations. The number of vary on operations the line had in the collection interval.

Protocol. Protocol type. This will be set to I for IDLC.

Reserved. An ignored field.

Receive CRC errors. The number of received frames that contain a cyclic-redundancy-check (CRC) error. This indicates that the data was not received error-free.

Sequence errors. The number of frames received during the time interval that contained sequence numbers indicating that frames were lost.

Short frame errors. The number of short frames received. A short frame is a frame that has fewer octets between its start flag and end flag than are permitted.

LAPD Format: For a description of the fields in this format, see "LAPD Field Descriptions" on page 51-16.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Network interface description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	Loss of frame alignment
56	38	BINARY(4)	Local end code violation
60	3C	BINARY(4)	Detected access transmission error in
64	40	BINARY(4)	Detected access transmission error out
68	44	BINARY(4)	Far end code violation
72	48	BINARY(4)	Errored seconds
76	4C	BINARY(4)	Severely errored seconds
80	50	BINARY(4)	Collision detect
84	54	BINARY(4)	Receive CRC errors
88	58	BINARY(4)	Short frame errors
92	5C	BINARY(4)	Aborts received
96	60	BINARY(4)	Sequence errors
100	64	BINARY(4)	Frames transmitted
104	68	BINARY(4)	Frames retransmitted
108	6C	BINARY(4)	Frames received
112	70	BINARY(4)	Frames received in error
116	74	BINARY(4)	Total outgoing calls
120	78	BINARY(4)	Retry for outgoing calls
124	7C	BINARY(4)	Total incoming calls
128	80	BINARY(4)	Retry for incoming calls
132	84	CHAR(1)	S1 maintenance channel
133	85	CHAR(63)	Reserved

LAPD Field Descriptions

Aborts received. The number of frames received that contained high-level data link control (HDLC) abort indicators. This indicates that the remote equipment ended frames before they were complete.

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be performed on this field.

Bytes received. The total number of bytes received from the remote link station. This includes no errors.

Bytes transmitted. The total number of bytes transmitted to a remote link station. This includes bytes retransmitted and bytes sent on transmissions stopped by transmit underrun, in addition to successful transmissions.

Collision detect. The number of times the terminal equipment (TE) detected that its transmitted frame has been corrupted by another TE attempting to use the same bus.

Detected access transmission error in. The number of times the network termination 1 (NT1) notified the terminal equipment (TE) of an error in data crossing the U interface from the line transmission termination (LT) to the NT1. The NT1 reports the errors to the TE through the maintenance channel S1.

Detected access transmission error out. The number of times the network termination 1 (NT1) notified the terminal equipment (TE) of an error in data crossing the U interface from the NT1 to the line transmission termination (LT). The NT1 reports the errors to the TE through the maintenance channel S1.

Errored seconds. The number of seconds that had at least one DTSE-in or DTSE-out error. DTSE is a maintenance channel that measures the quality of the line.

Far end code violation. The number of unintended code violations detected by the network termination 1 (NT1), and counted by the terminal equipment (TE), for frames transmitted to the NT1 on the interface for the T reference point. The NT1 reports a violation to the TE through the maintenance channel S1.

Frames received. The total number of information (I), unnumbered information (UI), and supervisory (S) frames received from the remote link station. This includes no errors.

Frames received in error. The sum of receive CRC errors, short frame errors, overrun, underrun, aborts received, and frame sequence errors.

Frames retransmitted. The number of frames requiring retransmission due to errors. Errors can be caused by a remote device that is failing or cannot receive data fast enough.

Frames transmitted. The total number of information (I), unnumbered information (UI), and supervisory (S) frames sent to a remote link station. This includes frames retrans-

mitted and frames sent on transmissions stopped by transmit underrun, in addition to successful transmissions.

IOP address. The IOP bus address. This number can range from 0 through 31. No delta calculation should be performed on this field.

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Local end code violation. This condition is counted by the terminal equipment (TE) to indicate unintended code violation detected by the TE for frames received at the interface for the ISDN S/T reference point.

Loss of frame alignment. The total number of times when a time period equivalent to two 48-bit frames has elapsed without having detected valid pairs of line code violations.

Network interface description. The name of the network interface description.

Number of vary on operations. The number of vary on operations the line had in the collection interval.

Protocol. Protocol type. This will be set to D for LAPD.

Receive CRC errors. The number of frames received that contain a cyclic-redundancy-check (CRC) error.

Reserved. An ignored field.

Retry for incoming calls. The number of incoming calls that were rejected by the network.

Retry for outgoing calls. The number of outgoing calls that were rejected by the network.

S1 maintenance channel. This field will be set to one if this ISDN had maintenance channel active.

Sequence errors. The number of received frames that contained sequence numbers that indicated frames were lost.

Severely errored seconds. The number of seconds that had more than three DTSE-in and DTSE-out errors.

Short frame errors. The number of short frames received. A short frame is a frame that has fewer octets between its start flag and end flag than are permitted.

Total incoming calls. The total number of incoming call attempts.

Total outgoing calls. The total number of outgoing call attempts.

SDLC Format: For a description of the fields in this format, see "SDLC Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	I-frames retransmitted
56	38	BINARY(4)	Error-free frames received
60	3C	BINARY(4)	Frames received in error
64	40	BINARY(4)	Invalid frames received
68	44	BINARY(4)	Link resets
72	48	BINARY(4)	I-frames transmitted
76	4C	BINARY(4)	Frames retransmitted
80	50	BINARY(4)	RR frames transmitted
84	54	BINARY(4)	RR frames received
88	58	BINARY(4)	RNR frames transmitted
92	5C	BINARY(4)	RNR frames received
96	60	BINARY(4)	Polling wait time
100	64	CHAR(96)	Reserved

SDLC Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be performed on this field.

Bytes received. The number of bytes received (data and control characters), including bytes received in error.

Bytes transmitted. The number of bytes transmitted (data and control characters) including bytes transmitted again because of errors.

Error-free frames received. The number of I-frames, supervisory frames, and frames not numbered that were

received without error (whether or not they were transmitted again from the remote side).

Frames received in error. The number of I-frames, supervisory frames, and frames not numbered that were received in error. The following are the error possibilities:

- A supervisory frame or I-frame was received with an Nr count that is requesting retransmission of a frame.
- An I-frame was received with an Ns count that indicates that frames were missed.
- A frame is received with a frame-check-sequence error, an abnormal end, a receive overrun, or a frame-truncated error.

Frames retransmitted. The number of I-frames, supervisory frames, and frames not numbered that were transmitted again.

I-frames retransmitted. The number of I-frames transmitted again.

I-frames transmitted. The number of I-frames transmitted.

IOP address. The IOP bus address. This number can range from 0 through 31. No delta calculation should be performed on this field.

Invalid frames received. The number of invalid frames received. These are frames received with either a short frame error (frame is less than 32 bits) or a residue error (frame is not on a byte boundary).

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Link resets. The number of times a set normal response mode (SNRM) was received when the station was already in normal response mode.

Number of vary on operations. The number of vary on operations the line had in the collection interval.

Polling wait time. The length of time (in tenths of seconds) that the system waits for the response to a poll while in normal disconnect mode before polling the next station. No delta calculation should be done on this field.

Protocol. Protocol type. This will be set to S for SDLC.

RNR frames received. The number of receive-not-ready supervisory frames received.

RNR frames transmitted. The number of receive-not-ready supervisory frames transmitted.

RR frames received. The number of receive-ready supervisory frames received.

RR frames transmitted. The number of receive-ready supervisory frames transmitted.

X.25 Format: For a description of the fields in this format, see "X.25 Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bus number
4	4	BINARY(4)	IOP address
8	8	CHAR(1)	Protocol
9	9	CHAR(10)	Line description
19	13	CHAR(1)	Line active
20	14	CHAR(12)	Reserved
32	20	BINARY(4)	Line speed
36	24	BINARY(4)	Number of vary on operations
40	28	BINARY(4)	Active time
44	2C	BINARY(4)	Bytes transmitted
48	30	BINARY(4)	Bytes received
52	34	BINARY(4)	I-frames retransmitted
56	38	BINARY(4)	Frames received in error
60	3C	BINARY(4)	Invalid frames received
64	40	BINARY(4)	Link resets
68	44	BINARY(4)	I-frames transmitted
72	48	BINARY(4)	Error-free frames received
76	4C	BINARY(4)	RR frames transmitted
80	50	BINARY(4)	RR frames received
84	54	BINARY(4)	RNR frames transmitted
88	58	BINARY(4)	RNR frames received
92	5C	BINARY(4)	Reset packets transmitted
96	60	BINARY(4)	Reset packets received
100	64	CHAR(96)	Reserved

X.25 Field Descriptions

Active time. The amount of time in seconds that the line was active (varied on). This field should be used instead of interval time for all time-dependent fields calculated (for example, line utilization) to get accurate statistics.

Bus number. The IOP bus number. This number can range from 0 through 7. No delta calculation should be performed on this field.

Bytes received. The number of bytes received (data and control characters), including bytes received in error.

Bytes transmitted. The number of bytes transmitted (data and control characters) including bytes transmitted again because of errors.

Error-free frames received. The number of I-frames, supervisory frames, and frames not numbered that were received without error (whether or not they were transmitted again from the remote side).

Frames received in error. The number of I-frames, supervisory frames, and frames not numbered that were received in error. The following are the error possibilities:

- A supervisory frame or I-frame was received with an Nr count that is requesting retransmission of a frame.
- An I-frame was received with an Ns count that indicates that frames were missed.
- A frame is received with a frame-check-sequence error, an abnormal end, a receive overrun, or a frame-truncated error.

I-frames retransmitted. The number of I-frames transmitted again.

I-frames transmitted. The number of I-frames transmitted excluding I-frames transmitted again.

IOP address. The IOP bus address. This number can range from 0 through 31. No delta calculation should be performed on this field.

Invalid frames received. The number of invalid frames received. These are frames received with either a short frame error (frame is less than 32 bits) or a residue error (frame is not on a byte boundary).

Line active. The state of the line when the collection interval ended. The values are:

0 The line is not active.
1 The line is active.

Line description. The name of the description for this line.

Line speed. The speed of this line in bits per second (bps). No delta calculation should be performed on this field.

Link resets. The number of times a set normal response mode (SNRM) was received when the station was already in normal response mode.

Number of vary on operations. The number of vary on operations the line had in the collection interval.

Protocol. Protocol type. This will be set to X for X.25.

Reserved. An ignored field.

Reset packets received. The number of reset packets received.

Reset packets transmitted. The number of reset packets transmitted.

RNR frames received. The number of receive-not-ready supervisory frames received.

RNR frames transmitted. The number of receive-not-ready supervisory frames transmitted.

RR frames received. The number of receive-ready supervisory frames received.

RR frames transmitted. The number of receive-ready supervisory frames transmitted.

Error Messages

CPF0A42 E Collector ended abnormally.
CPF0A43 E Data not available.
CPF0A44 E Collection not active for user.
CPF0A45 E Cannot copy data to user space &1.
CPF0A47 E User space &1 in lib &2 not large enough.
CPF24B4 E Severe error addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Work with Collector (QPMWKCOL) API

Parameters

Required Parameter Group:

1	Type of action to perform	Input	Char(10)
2	Type of resource	Input	Char(10)
3	Time between collections	Input	Binary(4)
4	Qualified user space name	Input	Char(20)
5	First sequence number	Output	Binary(4)
6	Error code	I/O	Char(*)

The Work with Collector (QPMWKCOL) API starts, ends, or changes the collection of performance data for a particular resource on your system. The API collects data similar to the performance monitor (Start Performance Monitor (STRPFRMON) command), but the performance collector used by the API has the following distinctions:

- It can collect data only for specific resources.
- It collects data at smaller time intervals.
- It deposits the collected data into user spaces, not database files. Because of this, the data exists only until the next collection (unless the user stores it).
- Multiple users can collect data at the same time.
- The same resource type data can be collected by different users with the same or different interval lengths.
- Different resources can be collected at different interval times.
- It will collect data until all collections have been explicitly ended or all of the user's jobs have ended.
- It does not calculate deltas on the data collected. For more information on deltas, see the List Performance Data (QPMLPFRD) API.

When the first user of the collector issues a call to the QPMWKCOL API, two jobs (QPMASERV and QPMACLCT) are submitted to batch. QPMASERV acts as a server, communicating between the APIs (QPMWKCOL and QPMLPFRD) and the QPMACLCT job, which does the actual data collection. These jobs run at priority 0 in subsystem QSYSWRK. No matter how many users are collecting data, there will only be one instance of each job running. The programs will continue to run until all users have ended all of their collections. They will also end if none of the users' jobs are still active.

To start a data collection for a resource, call the QPMWKCOL API using the following:

- The value *START for the type of action to perform parameter
- The type of resource data to collect (job, pool, disk, input/output processor (IOP), or communications)
- The length of time between collections (15, 30, 60, 120, or 240 seconds)
- The name and library of the user space the data should be copied into
- The error code parameter

A separate request must be made for each resource desired. When the request is valid, the sequence number of the first collection will be passed back to the user. The sequence number is incremented for every second that has passed and can be used to see if an interval collection was missed. For example, if 15 is the first sequence number received back from the QPMWKCOL API but 30 is the sequence number received back from the List Performance Data (QPMLPFRD) API, you missed the collection of data with a sequence number of 15.

By using *CHANGE for the type of action to perform parameter, the interval time or the user space for a resource can be changed.

To end a collection, use *END for the type of action to perform parameter. Because a collection will continue to be active until it is ended, it is important that any collections that are no longer needed be ended. If an *END request is from the last user of a resource, data collection of the resource stops. If not, the resource will still be collected, but this user will no longer have access to data until a *START request is issued again. After the last user of the collector ends all of his collections, the collector jobs (QPMASERV and QPMACLCT) end.

Because QPMWKCOL works with the QPMLPFRD API, the parameters selected for QPMWKCOL will affect QPMLPFRD. For example, the interval time selected determines how often QPMLPFRD should be called. Because the new data replaces old data, if QPMLPFRD is not called before the next interval is collected, the data from the previous interval will be lost, although the deltas may still be calculated for the longer interval. The qualified user space is also an important parameter to QPMLPFRD because this is the space that QPMLPFRD will copy the performance data into. If the space is not large enough to hold all the data or if the space is locked, an error message will be issued to the user.

Authorities and Locks

User Space Authority *USE
Library Authority *USE
User Space Lock *EXCLRD

Required Parameter Group

Type of action to perform
INPUT; CHAR(10)
Whether you want to start, end, or change the collection of a resource. The following values may be specified:

***START**	Start the collection of the specified resource.
***END**	End the collection of the specified resource.
***CHANGE**	Change the collection of the specified resource.

Type of resource
INPUT; CHAR(10)
The type of resource to start, end, or change. The following values may be specified.

***JOB**	Job-related information
***POOL**	Pool-related information
***DISK**	Disk-related information
***IOP**	IOP-related information
***COMM**	Communications-related information

Time between collections
INPUT; BINARY(4)
The number of seconds between each new collection of data. The following values may be specified.

15	Collect every 15 seconds.
30	Collect every 30 seconds.
60	Collect every 60 seconds.
120	Collect every 120 seconds (2 minutes).
240	Collect every 240 seconds (4 minutes).

Notes:

1. The disk- and IOP-related data require a minimum of 30 seconds between collections.
2. The communication-related data requires a minimum of 60 seconds between collections.

Qualified user space name
INPUT; CHAR(20)
The name of the user space that is to receive the data for this type of resource. The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The special values for the library name are:

***LIBL**	Library list is used.
***CURLIB**	Current library is used.

If no library is specified as the current library for the job, QGPL is used. Both entries are left-justified.

First sequence number
OUTPUT; BINARY(4)
The sequence number of the first data collection that will be available for the user.

Error code
I/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

CPF0A37 E Request type &1 not valid.
CPF0A38 E Resource type &1 not valid.
CPF0A39 E Interval time of &1 seconds not valid.
CPF0A40 E Interval time of &1 seconds for IOP data not valid.
CPF0A41 E &1 seconds for communications data not valid.
CPF0A42 E Collector ended abnormally.
CPF0A44 E Collection not active for user.
CPF0A45 E Cannot copy data to user space &1.
CPF0A46 E Interval time of &1 seconds for disk data not valid.
CPF0A47 E User space &1 in lib &2 not large enough.
CPF24B4 E Severe error addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Performance Monitor Exit Program

Parameters

Required Parameter Group:

1	Member name	Input	Char(10)
2	Library name	Input	Char(10)

The Performance Monitor exit program is called to process the performance data just collected by the performance monitor. You would write an exit program for the performance monitor if you wanted to be sure that the performance data being collected was processed as soon as the monitor was done collecting it.

Note: The Performance Monitor exit program pertains to the Start Performance Monitor (STRPFRMON) command, not the Performance Monitor Collector APIs.

Required Parameter Group

Member name
INPUT; CHAR(10)
The performance data member name.

Library name
INPUT; CHAR(10)
The library that contains the performance data.

Error Messages

The performance monitor handles any error that could occur in the exit program. Not only does the performance monitor contain generic MCH (machine), CPF (OS/400), and PFR (performance) error messages, but it also contains a function-check handler.

Part 18. Program and CL Command APIs

Chapter 52. Program and CL Command APIs

This chapter describes the APIs that you can use to create programs, to retrieve program information, and to retrieve command information. The APIs are:

Create Program (QPRCRTPG) converts the symbolic representation of a machine interface (MI) program into a program object.

List ILE Program Information (QBNLPGMI) gives information about Integrated Language Environment * (ILE *) programs, similar to the Display Program (DSPPGM) command.

List Service Program Information (QBNLSPGM) gives information about service programs, similar to the Display Service Program (DSPSRVPGM) command,

Process Commands (QCAPCMD) performs command analyzer processing on command strings.

Retrieve Command Information (QCDRCMDI) retrieves information from a command definition object and places it into a single variable in the calling program.

Retrieve Program Associated Space (QCLRPGAS) retrieves information from the associated space of a user-state, user-domain program.

Retrieve Program Information (QCLRPGMI) retrieves program information and places it into a single variable in the calling program.

Retrieve Service Program Information (QBNRSPGM) retrieves service program information and places it into a single variable in the calling program. This information is similar to the information returned using the Display Service Program (DSPSRVPGM) command.

Store Program Associated Space (QCLSPGAS) stores information in the associated space of a user-state, user-domain program.

Also included in this chapter are programming tips for using the QPRCRTPG API, which include:

- MI program syntax, including labels and instruction, declare, and directive statements
- MI program coding techniques, such as how to use declare statements, space objects, and constants

Before using this information, you should have some MI programming experience and understand the concepts in the *MI Functional Reference*.

Create Program (QPRCRTPG) API

Parameters

Required Parameter Group:

1	Intermediate representation of the program	Input	Char(*)
2	Length of intermediate representation of program	Input	Binary(4)
3	Qualified program name	Input	Char(20)
4	Program text	Input	Char(50)
5	Qualified source file name	Input	Char(20)
6	Source file member information	Input	Char(10)
7	Source file last changed date and time information	Input	Char(13)
8	Qualified printer file name	Input	Char(20)
9	Starting page number	Input	Binary(4)
10	Public authority	Input	Char(10)
11	Option template	Input	Char(*)
12	Number of option template entries	Input	Binary(4)

Optional Parameter:

13	Error code	I/O	Char(*)

The Create Program (QPRCRTPG) API converts the symbolic representation of a machine interface (MI) program into a program object. This symbolic representation is known as the intermediate representation of a program.

The QPRCRTPG API creates a program object that resides in the *USER domain and runs in the *USER state. If you want the program object to be temporary, you must do one of the following:

- Delete the object when you no longer need it.
- Create the object in the QTEMP library, and let the system delete the object automatically when the job ends.

You can specify program objects created with the QPRCRTPG API in CL commands that process objects of type *PGM. For example, you can:

- Save and restore program objects using the Save Object (SAVOBJ) and Restore Object (RSTOBJ) commands.
- Delete program objects using the Delete Program (DLTPGM) command.
- Run program objects using the Call (CALL) command.
- Rename program objects using the Rename Object (RNMOBJ) command.
- Move program objects to a different library using the Move Object (MOVOBJ) command.

Authorities and Locks

Program Authority

*ALL. Required only if the program already exists and the option value *REPLACE is specified.

Program Library Authority
*CHANGE
Printer File Authority
*USE
Printer File Library Authority
*USE

Required Parameter Group

Intermediate representation of the program
INPUT; CHAR(*)
A string containing the intermediate representation of the program to be processed by the QPRCRTPG API. See "Program Syntax" on page 52-6.

Length of intermediate representation of program
INPUT; BINARY(4)
The size, in bytes, of the intermediate representation of the program.

Qualified program name
INPUT; CHAR(20)
The name and library of the program to be created or replaced. The first 10 characters contain the program name, and the second 10 characters contain the name of the library where the program is located. The special value *CURLIB may be used for the library name.

Program text
INPUT; CHAR(50)
Text that briefly describes the program.

Qualified source file name
INPUT; CHAR(20)
The name and library containing the source program. The first 10 characters contain the source file name, and the second 10 characters contain the name of the library where the file is located. This places the value in the program object's service description. The special value *NONE may be used for the source file name. If you specify *NONE, no source file information is placed in the program object's service description. A special value, such as *LIBL, is not valid for the source file library.

Source file member information
INPUT; CHAR(10)
The file member containing the source program. This places the value in the program object's service description.

This value must be blanks if you specify *NONE as the **source file name**.

Source file last changed date and time information
INPUT; CHAR(13)
The date and time the member of the source file was last updated. The format of this field is in the CYYMMDDHHMMSS format, where:

C	Century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
YY	Year
MM	Month
DD	Day
HH	Hour
MM	Minute
SS	Second

This places the value in the program object's service description.

This value must be blank if you specify *NONE for the source file name parameter.

Qualified printer file name
INPUT; CHAR(20)
The name and library containing the printer file used to generate listings. The first 10 characters contain the printer file name, and the second 10 characters contain the name of the library where the file is located. The only special values supported for the library name are *LIBL and *CURLIB.

This value is ignored if you specify *NOLIST for the generate listing option (see "Values for the Option Template Parameter" on page 52-3).

Starting page number
INPUT; BINARY(4)
The first page number to be used on listings. This value should be between 1 and 9999; otherwise, the API uses 1.

This value is ignored if you specify *NOLIST for the generate listing option (see "Values for the Option Template Parameter" on page 52-3).

Public authority
INPUT; CHAR(10)
The authority you give the users who do not have specific private authorities to the object, and where the user's group has no specific authority to the object. The values allowed are:

*CHANGE
*ALL
*USE
*EXCLUDE
The name of an authorization list

Option template
INPUT; CHAR(*)
This is an array of options. You can specify between 0 and 16 values. Each entry contains a CHAR(11) value as described in "Values for the Option Template Parameter."

Number of option template entries
INPUT; BINARY(4)
The number of option template entries. The value must be between 0 and 16.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Values for the Option Template Parameter

When you are using the QPRCRTPG API, you can specify a value in the option template. Only one value per option should be specified. If you specify more than one, the system only uses the first one. If you specify no value for a given option, the system uses the default value (underlined).

Create program object
Creates a program object.

The values allowed are:

***GEN** Generates a program and places the program in the appropriate library.
***NOGEN** No program is generated. The syntax of the intermediate representation of the program is checked, and if the generate listing option is *LIST, a listing is produced.

Replace program
Replaces the existing program if a program by the same name already exists in the specified library.

The values allowed are:

***NOREPLACE** Does not replace an existing program by the same name in the specified library.
***REPLACE** Replaces the existing program by moving it to the QRPLOBJ library.

Generate listing
Generates an output listing.

The values allowed are:

***NOLIST** Does not generate a listing.
***LIST** Generates a listing. You must specify the following parameters:
- Printer file name and library
- Starting page number

Create cross-reference listing
Whether the listing is to contain a cross-reference list of variable and data item references.

The values allowed are:

***NOXREF** Does not create cross-reference listing.
***XREF** Creates a cross-reference listing of references to variables, labels, or both.

Create summary listing
Whether the listing is to contain a list of program attributes.

The values allowed are:

***NOATR** Does not create a summary listing section.
***ATR** Creates a summary listing section.

User profile
The values allowed are:

***USER** The user profile of the user running the program is used as a source of authority when this program runs.
***ADOPT** When the program runs, the object authority of both the program's owner and user are used.
***OWNER** The system uses the user profile of the owner of the program as a source of authority when this program runs. Programs called by this program adopt this authority.

Use adopted authority
Whether the system uses the program-adopted authority from the calling programs as a source of authority when this program is running.

The values allowed are:

***ADPAUT** The system uses program-adopted authority from the calling program.
***NOADPAUT** The system does not use program-adopted authority from the calling program.

Constrain arrays
The values allowed are:

***SUBSCR** Constrains arrays. This requests additional run-time checks to ensure that references to array elements are not outside the bounds of the declare statement. This option causes the resulting program to run slower.
***NOSUBSCR** Does not constrain arrays. The results of references to array elements outside the bounds of the declare statement are not defined.
***UNCON** Allows fully unconstrained arrays. This ensures that references to array elements outside the bounds of the declare statement act as if the array element actually exists.

Note: This program attribute may be changed at run-time using the Override Program Attributes (OVRPGATR) MI instruction.

Constrain strings
The values allowed are:

***SUBSTR** Constrains strings. This requests additional run-time checks to ensure that references to character strings are not outside the bounds of the declare statement. This option causes the resulting program to run slower.
***NOSUBSTR** Does not constrain strings. The results of substring references outside the bounds of the declare statement are not defined.

Note: You can change this program attribute at run-time using the Override Program Attributes (OVRPGATR) MI instruction.

Initialize static storage

Static storage is allocated the first time a program is called. It remains allocated until explicitly deallocated.

The values allowed are:

***CLRPSSA** Initializes static storage. This code clears the program static storage area (PSSA) on entry using the Call External (CALLX) MI instruction.

***NOCLRPSSA** Does not initialize the PSSA.

Initialize automatic storage

Automatic storage is allocated each time a program runs and automatically deallocated when no longer needed.

The values allowed are:

***CLRPASA** Initializes automatic storage. This code clears the program automatic storage area (PASA) on entry using the Call External (CALLX) MI instruction.

***NOCLRPASA** Does not initialize the PASA.

Ignore decimal data errors

Whether errors found in decimal data result in exceptions.

The values allowed are:

***NOIGNDEC** Does not ignore decimal data errors.

When you specify *NOIGNDEC, decimal values used in numeric operations are checked for valid decimal digits and sign codes. If the system finds an error, it signals an exception.

***IGNDEC** Ignores data decimal errors.

When you specify *IGNDEC, decimal values used in numeric operations ensure they contain valid decimal digit and sign codes. However, the system treats digits that are not valid as zeros and signs that are not valid as positive signs. There is no exception signaled.

This option applies to only a subset of the numeric operations you specify.

Note: In all cases, the system signals decimal data errors if you use data pointers to address any of the instruction's operands.

The following list contains the MI instructions this option affects:

MI Instruction	Packed Source Operands Supported	Zoned Source Operands Supported	Notes
ADDN		X	
CMPNV		X	
CVTCN	X	X	You must specify operand 3 (the numeric view to be used for operand 2) as a constant and no data-pointer-defined operands.
CVTDFFP		X	
CVTNC	X	X	You must specify operand 3 (the numeric view to be used for operand 1) as a constant and no data-pointer-defined operands.
CPYNV	X	X	You must specify no data-pointer-defined operands.
DIV		X	
DIVREM		X	
EDIT		X	You must specify no data-pointer-defined operands.
EXTRMAG		X	
MULT		X	
NEG		X	
REM		X	
SCALE		X	
SUBN		X	

When you specify *IGNDEC, the system may still signal the decimal data exception. That is, other MI instructions and instruction combinations not listed above may signal the decimal data exception when the system finds decimal data that is bad.

Ignore binary data size errors

The values allowed are:

***NOIGNBIN** The system handles binary data size errors normally. When a binary size error occurs, an exception is signaled and the receiver contains the left-truncated result.

***IGNBIN** The system ignores binary data size errors. This is used when an overflow or underflow occurs on a computation and when a control MI instruction has a receiver that is a binary field. The receiver contains the left-truncated result.

Support coincident operands

The system overlaps coincident operands between the source and receiver operands in one or more program instructions. **Coincident operands** are operands that overlap physically, in storage.

The values allowed are:

***NOOVERLAP** Does not support coincident operands. If you specify *NOOVERLAP, you guarantee that coincident operand overlap will not occur while running the instruction. Therefore, the system can use the

receiver on an instruction as a work area during operations performed to produce the final result. Using the receiver as a work area does not use as much processor resource as would be required to move the final result from an internal work area to the receiver.

***OVERLAP** Supports coincident operands. If you specify *OVERLAP, the operands on an MI instruction may overlap. Therefore, the system cannot use the receiver on an instruction as a work area during operations that produce the final result. This can require more processor resource for running the instruction but it ensures valid results if an overlap occurs.

The following is a list of instructions this option affects:

- Add logical character (ADDLC)
- Add numeric (ADDN)
- And (AND)
- Compute math function using one input value (CMF1)
- Concatenate (CAT)
- Convert character to numeric (CVTCN)
- Convert decimal form to floating-point (CVTDFFP)
- Convert external form to numeric value (CVTEFN)
- Convert floating-point to decimal form (CVTFPDF)
- Convert numeric to character (CVTNC)
- Copy bytes left adjusted with pad (CPYBLAP)
- Copy bytes right adjusted with pad (CPYBRAP)
- Divide (DIV)
- Divide with remainder (DIVREM)
- Exclusive or (XOR)
- Multiply (MULT)
- Or (OR)
- Remainder (REM)
- Scale (SCALE)
- Subtract logical character (SUBLC)
- Subtract numeric (SUBN)
- Trim length (TRIML)

Allow duplicate declares

The values allowed are:

***NODUP** This does not allow a program object to be declared more than once. This requests that duplicate declare (DCL) statements be diagnosed as errors.

***DUP** This allows a program object to be declared more than once. This requests that program objects declared more than once be pooled and not be diagnosed as errors.

Optimize

The values allowed are:

***OPT** This optimizes the program. In most instances, this produces the smallest and best running program. Occasionally, the source program may signal a MCH2802 escape message during processing. If this occurs, you should not optimize the program.

***NOOPT** This does not optimize the program. This requests the normal level code optimization when you create the program.

Error Messages

CPD0078 E Value &3 for parameter &2 not a valid name.
CPF2143 E Cannot allocate object &1 in &2 type *&3.
CPF2144 E Not authorized to &1 in &2 type *&3.
CPF2146 E Owner of new program and existing program not the same.
CPF2283 E Authorization list &1 does not exist.
CPF3CF1 E Error code parameter not valid.
CPF3C35 E Value &3 for parameter &2 not a valid name.
CPF3C5A E Number of option template entries is not valid.
CPF3C5B E Option template entry is not valid.
CPF3C5C E Source file name and library is not valid.
CPF3C5D E Source file member is not valid.
CPF3C5F E Internal Representation of Program (IRP) string length parameter is not valid.
CPF3C50 E Program &1 not created.
CPD0078 D Value &3 for parameter &2 not a valid name.
CPD3C50 D Internal Representation of Program (IRP) string length parameter is not valid.
CPD3C52 D Number of option template entries is not valid.
CPD3C53 D Option template entry is not valid.
CPD3C54 D Source file name and library is not valid.
CPD3C55 D Source file member is not valid.
CPD3C56 D Source file last changed date and time is not valid.
CPF3C56 E Source file last changed date and time is not valid.
CPF3C60 E Program name and library is not valid.
CPF3C61 E Authority is not valid.
CPF3C62 E Source file library specified.
CPF3C63 E Source file member specified.
CPF3C64 E Source file last changed date and time specified.
CPF6301 E Intermediate representation of program (IRP) contains &1 errors. Probable compiler error.
CPF6303 E Message &1, &2 received while running create program command.
CPF6304 E Library &1 not found.
CPF6306 E Program &1 in library &2 already exists.
CPF6307 E Program template value at offset &1, bit &2, length &3 not valid.
CPF6308 E Not authorized to create program.
CPF6309 E Not authorized to library &1.
CPF6455 E Member &2 file &1 in library &3 not found.
CPF6457 E Cannot allocate library &1 for program insertion.
CPF6551 E Work space &2 cannot be extended. Probable compiler error.

CPF6552 E	Space &2 type &3 subtype &4 not PRM work space.
CPF6553 E	PRM permanent table resolution failed. Probable compiler error.
CPF6554 E	Type of IST object &4 at offset &3 not valid. Probable PRM error.
CPF6555 E	Addressability field type not valid for IST number &4 at offset &3. Probable PRM error.
CPF6557 E	Error condition for IST &4 at &3 of IST space not valid. Probable PRM error.
CPF6560 E	Operation code &5 in MI instruction &3 at offset &6 not found in QPRODT.
CPF6561 E	Operand &4 in &3 at offset &5 in program template not valid.
CPF6563 E	Program was too large to be created.
CPF6564 E	Machine storage limit violation.
CPF6565 E	User profile storage limit exceeded.
CPF9872 E	Program &1 in library &2 ended. Reason code &3.

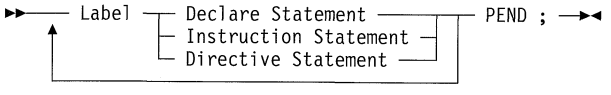
Program Attributes

The QPRCRTPG API creates programs that have the following attributes:

- An associated space of 0 bytes. You can use the QCLSPGAS API to store information in the program's associated space.
- Observability.
- A blank extended attribute.

Program Syntax

A program object consists of an **instruction stream** and an **object definition table (ODT)**. The intermediate representation of a program defines both of these components. It consists of one or more statements:



Instruction statements define MI instructions placed in the instruction stream. Declare statements define program objects placed in the ODT. Directive statements:

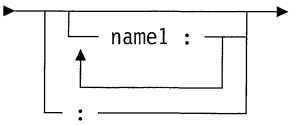
- Control the formatting of the output listing.
- Define entry-point program objects.
- Define symbolic breakpoints.
- Specify the end of the program.

The following sections explain how to define these statements.

Note: In the diagrams below, names that begin with an uppercase letter identify values specified in another diagram. Names that begin with a lowercase letter identify values defined in the table below the diagram.

Label

The following diagram and table show the possible labels:



Each name specified in the label generates a branch-point program object corresponding to the next MI instruction.

Constant	Range	Description
name1	Any	Label name for next instruction

Declare Statement

Declare statements define program data objects. All the declare statements in a program build the object definition table (ODT).

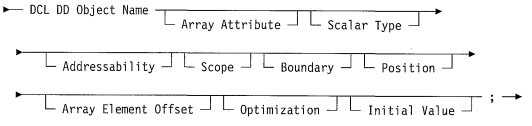
You cannot specifically declare branch and entry-point program objects. However, you can declare branch-point program objects using labels. You can also declare entry-point program objects using the entry directive statement.

The types of declare statements follow:

- Scalar Data Object Declare Statement
- Pointer Data Object Declare Statement
- Space Pointer Machine Object Declare Statement
- Operand List Declare Statement
- Instruction Definition List Declare Statement
- Exception Description Declare Statement
- Space Object Declare Statement
- Constant Object Declare Statement

Scalar-Data-Object Declare Statement

The following diagram and table show the scalar-data-object declare statement:

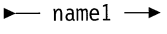


Only certain combinations of attributes are allowed based on the data object's addressability. The table below shows these combinations.

Addressability	Array Attribute	Array Element Offset	Position	Boundary	Initial Value
STAT	X		X		X
	X			X	X
AUTO	X		X		X
	X			X	X
DEF	X		X	X	X
	X	X	X	X	

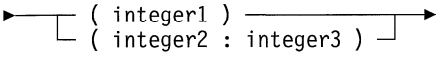
Addressability	Array Attribute	Array Element Offset	Position	Boundary	Initial Value
DIR	X		X	X	X
	X	X	X	X	
BAS	X		X	X	
BASPCO	X		X	X	
PARM	X			X	

Object Name: The following diagram and table show the possible object names:



Constant	Range	Description
name1	Any	Program object name to be declared

Array Attribute: The following diagram and table show the possible array attributes:

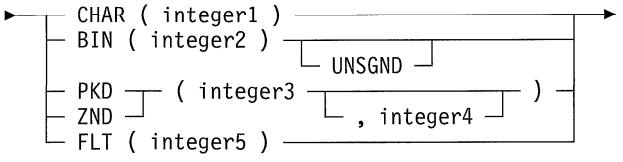


Constant	Range	Description
integer1	1 to 16 776 191	Dimension of the data object with an implied lower bound of 1.
integer2	-2 147 483 648 to 2 147 483 647	Lower bound of the array.
integer3	integer2 to 2 147 483 647	Upper bound of the array. The dimension (integer3 - integer2) cannot exceed 16 776 191.

Example: The following declare statements each define an array of 50 elements. The elements of ARRAY1 are numbered 1 to 50. The elements of ARRAY2 are numbered 0 to 49. Each element of the array is a BIN(2) field. The addressability of the arrays is static.

```
DCL DD ARRAY1(50) BIN(2);
DCL DD ARRAY2(0:49) BIN(2);
```

Scalar Type: The following diagram and tables show the possible data types of scalar items:

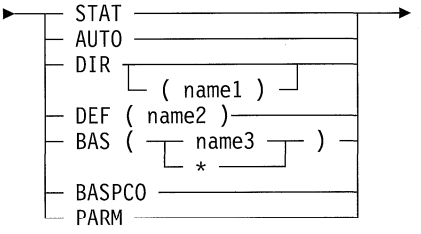


If you specify no value, the system uses BIN(2).

Keyword	Description
CHAR	Scalar type is a character string.
BIN	Scalar type is binary.
UNSGND	Scalar type is unsigned binary. If you do not specify this value, the scalar type is signed binary.
PKD	Scalar type is packed decimal.
ZND	Scalar type is zoned decimal.
FLT	Scalar type is floating-point.

Constant	Range	Description
integer1	See description.	Length in bytes of the character data object. If the data object is an array, the range is 1 to 32 767. Otherwise, the range is 1 to 16 776 191.
integer2	2 or 4	Length in bytes of the binary data object.
integer3	1 to 31	Total digits in the data object.
integer4	0 to integer3	Number of digits to the right of the assumed decimal point in the data object.
integer5	4 or 8	Precision in bytes of the data object.

Addressability: The following diagram and tables show the possible addressabilities:



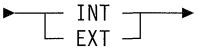
Keyword	Description
STAT	Addressability type is direct static.
AUTO	Addressability type is direct automatic.
DIR	Addressability type is defined. See "Using Space Objects" on page 52-16 for more information.
DEF	Addressability type is direct on the previous space.
BAS	Addressability type is based.
*	Object does not have explicit basing object.
BASPCO	Addressability type is based on process communication object space pointer.
PARM	Addressability type is a parameter.

Constant	Range	Description
name1	Any	Space object name

Constant	Range	Description
name2	Any	Scalar data object name or pointer data object name
name3	Any	Pointer data object name or space pointer object name

If you specify no value, the system uses STAT.

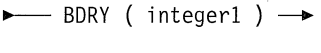
Scope: **Scope** refers to the ability to export a variable so that other programs can access it. The following diagram and table show the possible scopes:



Keyword	Description
INT	Data object is not externally accessible
EXT	Data object is externally accessible

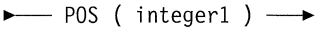
If you specify no value, the system uses INT.

Boundary: The following diagram and table show the possible boundaries:



Constant	Range	Description
integer1	1, 2, 4, 8, 16	Data object boundary

Position: The following diagram and table show the possible positions:



Constant	Range	Description
integer1	1 to 16 776 191	Data object position

Example: The following declare statements show how POS can be used along with DEF to access the same storage space in different ways:

```
DCL DD DATETIME CHAR(12);
DCL DD DATE CHAR(6) DEF(DATETIME);
DCL DD TIME CHAR(6) DEF(DATETIME) POS(7);
```

DATETIME represents a 12 character time and date stamp. The first 6 characters contain the date and the second 6 characters contain the time.

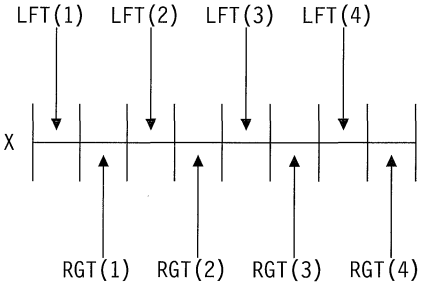
Array Element Offset: The following diagram and table show the possible array element offsets:

Constant	Range	Description
integer1	1 to 32767	Array element offset

Example: The following example shows AEO used in conjunction with DEF and POS:

```
DCL DD X CHAR(16);
DCL DD LFT(4) BIN(2) DEF(X) AEO(4) POS(1);
DCL DD RGT(4) BIN(2) DEF(X) AEO(4) POS(3);
```

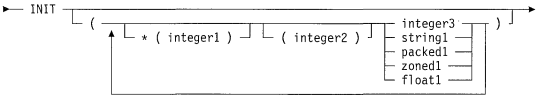
Both LFT and RGT redefine the storage declared by X. Because the size of each array element is smaller than the array element offset, there are 2-byte gaps between each array element:



Optimization: **Optimization** determines whether or not an item can be moved to a register and stored there over time. The following diagram shows the possible optimization:

This value indicates that the data object contains an abnormal value. You cannot optimize the value for more than a single reference because the value may be changed in a manner that the QPRCRTPG API cannot detect.

Initial Value: The following diagram and table show each possible initial value:



Constant	Range	Description
integer1	1 to 16 776 191	Position of elements in a character string
integer1	-2 147 483 648 to 2 147 483 647	Position of elements in an array
integer2	1 to 16 776 191	Replication factor in a character string or array
integer3	Any	Initial value for signed and unsigned binary data objects
string1	Any	Initial value for character string data objects
packed1	Any	Initial value for packed decimal data objects
zoned1	Any	Initial value for zoned decimal data objects
float1	Any	Initial value for floating-point data objects

Example: The following declare statement declares and initializes a 10-element array:

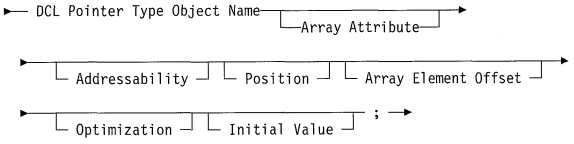
```
DCL DD IV(10) BIN(2) STAT INIT((1)10,*(2)(2)11,*(4)(3)12,*(7)(4)13);
```

There are four initial value elements. The following table describes this function:

Initial Value Element	Result	Position	Replication Factor	Initial value
(1)10	IV(1)=10	1 (default)	1	10
*(2)(2)11	IV(2)=11 IV(3)=11	2	2	11
*(4)(3)12	IV(4)=12 IV(5)=12 IV(6)=12	4	3	12
*(7)(4)13	IV(7)=13 IV(8)=13 IV(9)=13 IV(10)=13	7	4	13

Pointer-Data-Object Declare Statement

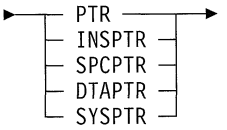
The following diagram and table show the pointer-data-object declare statement:



The system only allows certain combinations of attributes based on the data object's addressability. These combinations are listed as follows:

Addressability	Array Attribute	Array Element Offset Attribute	Position Attribute	Initial Value Attribute
STAT	X		X	X
AUTO	X		X	X
DEF	X	X	X	
			X	X
DIR	X	X	X	
			X	X
BAS	X		X	
BASPCO	X		X	
PARM	X			

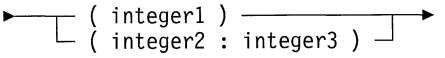
Pointer Type: The following diagram and table show the possible pointer types:



Keyword	Description
PTR	Pointer type is not specified.
INSPTR	Pointer type is the instruction pointer.
SPCPTR	Pointer type is the space pointer.
DTAPTR	Pointer type is the data pointer.
SYSPTR	Pointer type is the system pointer.

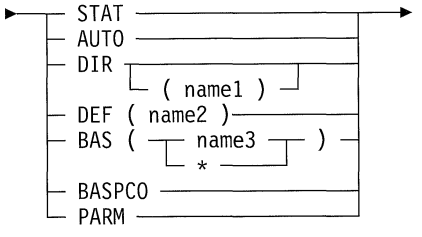
If you specify an initial value, you must specify INSPTR, SPCPTR, DTAPTR or SYSPTR.

Array Attribute: The following diagram and table show the possible array attributes:



Constant	Range	Description
integer1	1 to 1 000 000	Dimension of the data object with an implied lower bound of 1.
integer2	-2 147 483 648 to 2 147 483 647	Lower bound of the array.
integer3	integer2 to 2 147 483 647	Upper bound of the array. The dimension (integer3 - integer2) should not exceed 1 000 000.

Addressability: The following diagram and tables show the possible addressabilities:

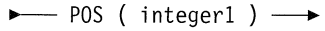


Keyword	Description
STAT	Addressability type is direct static.
AUTO	Addressability type is direct automatic.
DIR	Addressability type is defined. See "Using Space Objects" on page 52-16 for more information.
DEF	Addressability type is defined.
BAS	Addressability type is based.
*	Object does not have explicit basing object.
BASPCO	Addressability type is based on the process communication object space pointer.

Keyword	Description
PARM	Addressability type is parameter.

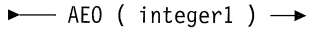
Constant	Range	Description
name1	Any	Space object name
name2	Any	Scalar data object name or the pointer data object name
name3	Any	Pointer data object name or the space pointer machine object name

Position: The following diagram and table show the possible positions:



Constant	Range	Description
integer1	1 to 16 776 191	Data object position

Array Element Offset Value: The following diagram and table show the possible array element offset values:

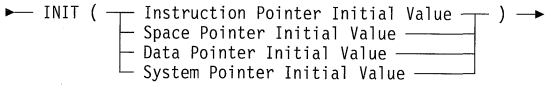


Constant	Range	Description
integer1	1 to 32 767	Array element offset

Optimization: The following diagram shows the possible optimizations:

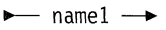
This value indicates that the data object contains an abnormal value. The system cannot optimize a value for more than a single reference because the value may be changed in a manner the QPRCRTPG API cannot find.

Initial Value: The following diagram shows each possible initial value:



An initial value can only be specified if a pointer-type value other than PTR is specified. The syntax of the initial value is based on the pointer-type value that was used.

Instruction Pointer Initial Value: The following diagram and table show the possible initial value for the instruction pointer:

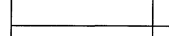


Constant	Range	Description
name1	Any	Label name

Example: The following statement declares and initializes an instruction pointer:

```
LABELI:

  :
  :

  DCL INSPTR INSTRUCTION_PTR INIT(LABELI);
```

Space Pointer Initial Value: The following diagram and table show the initial value for the space pointer:



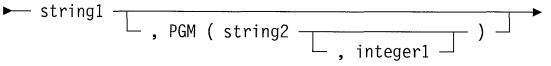
Constant	Range	Description
name1	Any	Scalar data object name or pointer data object name

Example: The following statement declares and initializes a space pointer:

```
DCL PTR ANY_POINTER;
DCL SPCPTR SPACE_PTR INIT(ANY_POINTER);
```

The pointer SPACE_PTR is initialized to point to the space location containing ANY_POINTER. It does *not* contain the value of ANY_POINTER.

Data Pointer Initial Value: The following diagram and table show the initial value for the data pointer:



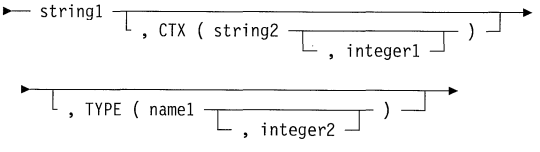
Constant	Range	Description
string1	32 bytes	External data object name
string2	30 bytes	Program containing external data object
integer1	0 to 255	Subtype of program

Example: The following statement declares and initializes a data pointer:

```
DCL DTAPTR DVALUE INIT("DBINARY",PGM("DPGM"));
```

The pointer DTAPTR refers to the externally defined program object DBINARY contained in program DPGM.

System Pointer Initial Value: The following diagram and tables show the initial value for the system pointer:



Constant	Range	Description
string1	1 to 30 bytes	System object
string2	1 to 30 bytes	Context where the system object is located
integer1	0 to 255	Subtype of the context

Constant	Range	Description
name1	See table below.	Symbolic type of the system object
integer2	0 to 255	Subtype of the system object

The following system object types are supported:

Type	Description
PGM	Program
CTX	Context
Q	Queue
SPC	Space
PCS	Process control space

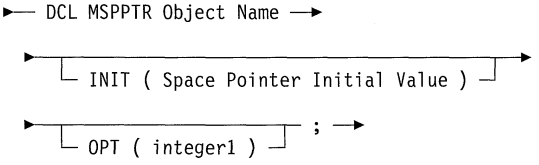
Example: The following statement declares and initializes a system pointer:

```
DCL SYSPTR SYSTEM_PTR INIT("MYPGM",CTX("PGMLIB"),TYPE(PGM));
```

The pointer SYSTEM_PTR refers to the *PGM object MYPGM in the PGMLIB library.

Space-Pointer-Machine-Object Declare Statement

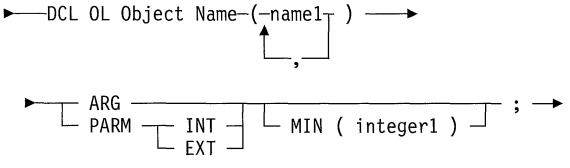
The following diagram and table show the space-pointer-machine-object declare statement:



Constant	Range	Description
integer1	0 to 255	Optimization priority value

Operand-List Declare Statement

The following diagram and tables show the operand-list declare statement:



Keyword	Description
ARG	Defines the argument list
PARM	Defines the parameter list
INT	An internal parameter list
EXT	An external parameter list

Constant	Range	Description
name1	Any	Scalar data object or a pointer data object name. Up to 255 names can be specified.
integer1	0 to 255	Minimum number of elements that the list can contain. This implicitly defines a variable-length operand list. If you do not specify the operand list, the system defines a fixed-length operand list. Up to 255 names can be specified.

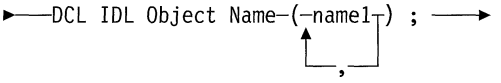
Example: The following statements declare both argument and parameter operand lists along with the associated argument and parameter data objects:

```
DCL DD ARG1 BIN(2);
DCL DD ARG2 CHAR(3);
DCL OL ARGUMENT_LIST (ARG1, ARG2) ARG;

DCL DD PARM1 BIN(2) PARM;
DCL DD PARM2 CHAR(3) PARM;
DCL OL PARAMETER_LIST (PARM1, PARM2) PARM EXT;
```

A parameter operand list that refers to the data objects has parameter (PARM) addressability.

Instruction-Definition-List Declare Statement

The following diagram and table show the instruction-definition-list declare statement:



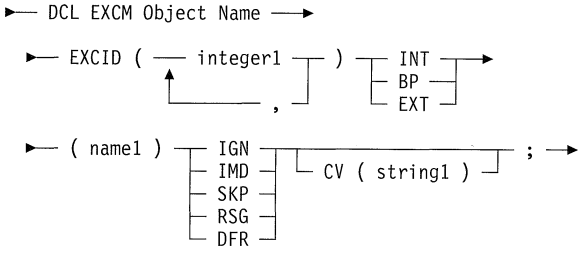
Constant	Range	Description
name1	Any	Label name. Up to 255 names can be specified.

Example: The following statements declare and use an instruction definition list:

```
LABEL1:

  :
  :

  DCL IDL INSTRUCTION_LIST (LABEL1,LABEL2,LABEL3);

  :
  :

LABEL2:
  B INSTRUCTION_LIST(3); /* Branch to LABEL3 */

  :
  :

LABEL3:
```

Exception-Description Declare Statement

The following diagram and tables show the exception-description declare statement:

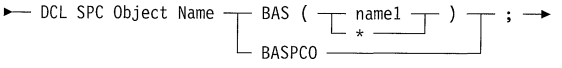


Keyword	Description
INT	Exception handler type is the internal entry point.
BP	Exception handler type is the internal branch point.
EXT	Exception handler type is the external entry point.
IGN	Exception handling action ignores any exceptions and continues processing.
IMD	Exception handling action passes control to the specified exception handler.
SKP	Exception handling action is to continue to search for another exception description to handle the exception.
RSG	Exception handling action continues to search for an exception description by signaling the exception again to the previous call.
DFR	Exception handling action postpones handling and saves exception data for later exception handling.

Constant	Range	Description
integer1	0 to 65535	Exception identifier
name1	Any	Name of the label for branch point exception handlers, name of the entry point for the internal exception handlers, and the name of the system pointer for the external exception handlers
string1	1 to 32 bytes	Compare value

Space-Object Declare Statement

The following diagram and tables show the space-object declare statement:



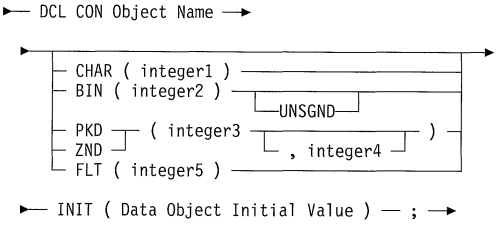
Keyword	Description
BAS	Addressability type is based.
*	Object does not have explicit basing object.
BASPCO	Addressability type is based on process communication object space pointer.

Constant	Range	Description
name1	Any	Basing pointer name for the space

For information on using space objects, refer to "Using Space Objects" on page 52-16.

Constant-Object Declare Statement

The following diagram and tables show the constant-object declare statement:



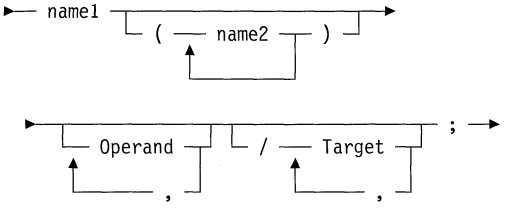
Keyword	Description
CHAR	Scalar type is character string.
BIN	Scalar type is binary.
UNSGND	Scalar type is unsigned binary.
PKD	Scalar type is packed decimal.
ZND	Scalar type is zoned decimal.
FLT	Scalar type is floating-point.

Constant	Range	Description
integer1	1 to 32 767	Length in bytes of the character data object
integer2	2 or 4	Length in bytes of the binary data object
integer3	1 to 31	Number of decimal digits
integer4	0 to integer3	Number of fractional digits
integer5	4 or 8	Number of bytes in floating-point constant

If you do not specify a scalar type, the system uses BIN(2).

Instruction Statement

An instruction statement defines an MI instruction. The instruction stream used to create the program is made up of all the instruction statements in the intermediate representation of the program.



Constant	Range	Description
name1	See description.	Opcode for this instruction, as defined in the *MI Functional Reference*.
name2	S, R, B, I	This is the form of the instruction. **S** Short **R** Round **B** Branch **I** Indicator

For the semantic meanings and the syntax restrictions (number and types of operands, optional forms, and so on) for individual MI instructions, see the *MI Functional Reference*.

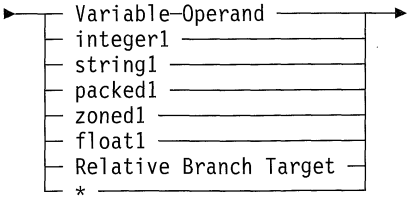
Following the abbreviated instruction name, you can specify the optional forms of certain MI instructions using a string of characters enclosed in parentheses. The following is an example of some of the various combinations possible for a single MI instruction, ADD NUMERIC:

```
ADDN      A,B,C;                Add numeric (A=B+C)
ADDN(S)   A,B;                  Add numeric short (A=A+B)
ADDN(SR)  A,B;                  Add numeric short and round (A=A+B)
ADDN(SB)  A,B/POS(X),NEG(Y);    Add numeric short and branch  (A=A+B,
                                branch to X if A>0, branch to Y if A<0)
ADDN(RI)  A,B,C/POS(I),NEG(J); Add numeric round and indicator (A=B+C;
                                I='on' if A>0; j='on' if A<0 )
```

Also note that the order of characters in the optional form string is not significant. Thus, all of the following instructions are both valid and equivalent:

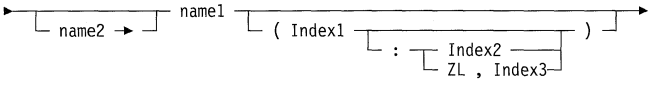
```
ADDN(SRB)A,B/POS(X);     Add numeric short, round and branch
ADDN(SBR)A,B/POS(X);     Add numeric short, round and branch
ADDN(RSB)A,B/POS(X);     Add numeric short, round and branch
```

Operand: The following diagram and table show the possible operands:



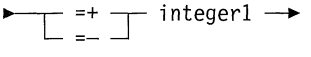
Constant	Range	Description
integer1	Any	Numeric binary scalar operand
string1	Any	Character scalar operand
packed1	Any	Numeric packed decimal scalar operand
zoned1	Any	Numeric zoned decimal scalar operand
float1	Any	Numeric floating-point scalar operand (4 or 8 bytes)
*		Null operand

Variable Operand: The following diagram and table show the possible variable operands:



Constant	Range	Description
name1	Any	Data object name to be used as a primary operand.
name2	Any	Pointer data object to be used as the basing pointer.
Index1	See description.	Subscript or substring start position. The range for array subscripts is between the lower bound of the array and the upper bound of the array. The range for substrings is between 1 and 16 776 191.
Index2	1 to 32 767	Length of the substring.
Index3	0 to 32 767	Length of the substring (zero allowed).

Relative Branch Target: The following diagram and table show the possible relative branch targets:



Constant	Range	Description
integer1	1 to 4095	Branch target instruction number *relative* to the current instruction. You must label the target (named or null label).

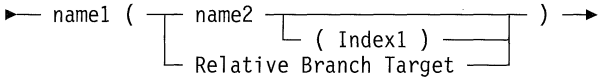
Note: You cannot use bianks between either the '=+' symbol set and integer1 or the '=-' symbol set and integer1. However, a blank must precede the symbol sets.

Example: The following instructions illustrate the use of relative branch targets:

```
  CPYNV X,0;
  CMPBLA(B) A,'1'/EQ( =+2);
  CPYNV X,1;
: CPYNV Y,X;        /* Destination of relative branch */
```

Note: A null label is placed in the destination instruction of the relative branch.

Target: The following diagram and table show the possible targets:



Constant	Range	Description
name1	See keyword table.	Keyword for branch or indicator forms. You can use an N before keywords to negate the condition except for IGN and DFR. See "Resultant Conditions", under each MI instruction for the valid values.
name2	Any	Label name, instruction pointer name, or instruction definition list name for the branch form. The name of character variable is for the indicator form.
Index1	1 to 255	Instruction definition list index. You can only specify this value when name2 is the name of an instruction definition list.

Notice that a null label has been placed on the destination instruction of the relative branch.

The following table shows the branch and indicator keywords:

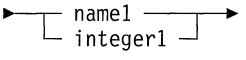
Keyword	Description
Group 1	
HI	High
MXD	Mixed
NOR	Normalized
POS	Positive
TR	Truncated record
ZC	Zero and carry
Group 2	
CR	Complete record
DEN	Denormalized
IGN	Exception ignored
LO	Low
NEG	Negative
NTZNTC	Not-zero and no carry
RO	Receiver overrun
Group 3	
AUTH	Authorized
DFR	Exception postponed
DQ	Dequeued
EQ	Equal
INF	Infinity
SE	Source all used
SGN	Signaled
ZER	Zero
ZNTC	Zero and no carry
Group 4	
EC	Escape code encountered
NAN	Not a number (NaN)
NTZC	Not-zero and carry
UNEQ	Unequal
UNOR	Unordered

By adding N to the beginning of the appropriate keyword you can form a not condition. For example, the code for "not equal" is NEQ.

All conditions coded on a particular instruction must be mutually exclusive. All conditions within a group are equivalent, and therefore, only one may be specified. For example, POS (positive) and HI (high) cannot be coded on the same instruction.

The not form of a condition is satisfied by any condition from another group. For example, NEQ (not equal) is satisfied by HI (high), LO (low), or UNOR (unordered). Therefore, you cannot specify NEQ with any of the other three. However, you can use NEQ and EQ (or any other keyword in group 3) together because they are mutually exclusive.

Index: The following diagram and table show the possible indexes:



Constant	Range	Description
name1	See description below.	Binary variable to use as the index
integer1	See description below.	Integer value to use as the index

An index is a numeric value that qualifies an array or substring reference. The context in which the index is used determines the range. For more information, refer to the preceding tables.

Directive Statements

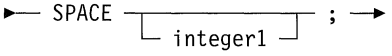
The directive statements are as follows:

- Title Directive Statement
- Space Directive Statement
- Eject Directive Statement
- Break Directive Statement
- Entry Directive Statement
- Reset Directive Statement
- Program End Directive Statement

Title Directive Statement: The title directive statement causes a heading to appear on the listings. Only one title directive statement may be specified in a program. The following diagram and table show the title directive statement:

Constant	Range	Description
string1	Any	Text of the title

Space Directive Statement: The space directive statement causes a blank line to appear in the listing. The following diagram and table show the space directive statement:



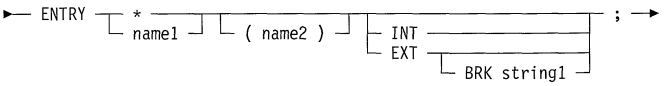
Constant	Range	Description
integer1	Any	Number of lines to skip

Eject Directive Statement: The eject directive statement causes the next line to appear on a new page. The following diagram shows the eject directive statement:

Break Directive Statement: The break directive statement allows symbolic breakpoints to be defined. The following diagram and table show the break directive statement:

Constant	Range	Description
string1	Any	Breakpoint name

Entry Directive Statement: The following diagram and tables show the entry directive statement:



Keyword	Description
INT	Internal entry point.
EXT	External entry point.
BRK	Symbolic breakpoint is associated with the entry point.
*	Entry point defined has no name or is associated with the next MI instruction.

Constant	Range	Description
name1	Any	Entry point name being defined
name2	Any	Parameter list name for this entry point
string1	1-10 bytes	Breakpoint name

The default scope is internal (INT).

The entry statement defines entry point program objects. The next instruction number is associated with this entry point. The entry statement is to be the definition point for this object, so the ODT number assigned to this object is the next available ODT number.

Reset Directive Statement: The following diagram shows the reset directive statement:

The specified name is a previously declared space object. The reset statement causes subsequent data object declarations containing the DIR attribute to use the specified space object. The system maintains next byte counts for each space object; these counts are not affected by the reset statement. For more information, see "Using Space Objects" on page 52-16.

Program End Directive Statement: The following diagram shows the program end directive statement:

This must be the last statement in the program. To ensure comments and strings end before processing the PEND statement, use the following statement:

```
/*'/*'/*"/*"*/; PEND;;;
```

Coding Techniques

This section contains additional information for coding the intermediate representation of a program.

Using Declare Statements

Use the following guidelines when using declare statements:

- A declare statement for data objects defined on another data object must occur after the declare statement for the data object on which it is defined.

  Example: The following sets of declare statements are valid:

  ```
  DCL DD A CHAR(5);
  DCL DD B CHAR(1) DEF(A);
  ```

  ```
  DCL DD A CHAR(5);
  DCL DD X BIN(2);
  DCL PTR P1 AUTO;
  DCL DD B CHAR(1) DEF(A);
  ```

  Example: The following declare statements are not valid because B is defined on A but is declared before A:

  ```
  DCL DD B CHAR(1) DEF(A);
  DCL DD A CHAR(5);
  ```

  This restriction also applies when there is a chain of dependencies.

  Example: In the figure below, B is defined on A and C is defined on B:

  ```
  DCL DD A CHAR(5);
  DCL DD B CHAR(3) DEF(A);
  DCL DD C CHAR(1) DEF(B);
  ```

  If any object in a chain of definitions, as shown in the previous examples, has an initial value specified, then the following restrictions apply:

  - No object in that chain can have the BAS (based) addressability attribute.
  - The highest level data object in the chain must be either static or automatic.
  - When you initialize the same area twice, the system uses the last value.

  Example: The following declare statements are valid because:

  - The BAS addressability attribute is not used.
  - Data object A (implicitly) has the static addressability attribute.

    ```
    DCL DD A CHAR(5);
    DCL DD B CHAR(3) DEF(A) INIT(C'YES');
    DCL DD C CHAR(1) DEF(B);
    ```

- All declare statements for the objects that make up the elements of an operand list must precede the declare statement for the operand list.
- When a declare statement for an exception description refers to a system pointer, the declare statement for the system pointer must precede the DCL for the exception description.

Using Space Objects

Space objects, when used in conjunction with program objects declared with the DIR attribute, provide a convenient way of declaring structures.

Note: Space objects, as used here, do not refer to OS/400 space objects.

When you declare a space object, a scalar data object with a scalar type of CHAR(32767) is created. This object contains the structure to be defined. Associated with this object is a "next byte" count. This value is initially 1 and represents the position where the next structure element will be placed.

Example: Simple Space Objects: After you declare a space object, you can declare one or more scalar or pointer data objects with an addressability attribute of DIR. As a result, the system automatically declares each object with the DEF and POS attributes. The name associated with the DEF attribute is the most recently declared space object. The value associated with the POS attribute is the space object's next byte count. After you declare the object, the system sets the next byte count associated with the space object to the next available position within the structure.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPC X BAS(PTR);          DCL DD X CHAR(32767) BAS(PTR);
DCL DD A CHAR(2) DIR;        DCL DD A CHAR(2) DEF(X) POS(1);
DCL DD B ZND(5,2) DIR;       DCL DD B ZND(5,2) DEF(X) POS(3);
DCL DD C FLT(4) DIR;         DCL DD C FLT(4) DEF(X) POS(8);
```

Example: Explicit Position Values: Data objects declared with DIR may also have an explicit POS value. The object is defined on the appropriate space object and uses the specified POS value. However, the next byte count is changed only if the POS value causes the count to increase.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPC X BAS(PTR);                DCL DD X CHAR(32767) BAS(PTR);
DCL DD A CHAR(4) DIR;              DCL DD A CHAR(4) DEF(X) POS(1);
DCL DD B CHAR(4) POS(20) DIR;      DCL DD B CHAR(4) DEF(X) POS(20);
DCL DD C CHAR(4) DIR;              DCL DD C CHAR(4) DEF(X) POS(24);
DCL DD D CHAR(4) POS(10) DIR;      DCL DD D CHAR(4) DEF(X) POS(10);
DCL DD E CHAR(4) DIR;              DCL DD E CHAR(4) DEF(X) POS(28);
```

Example: Explicit Boundary Alignment: When you declare objects with an explicit boundary other than 1, the object is positioned on the next available byte with that boundary. The position of any data object with the direct attribute is the next available byte in the space if no boundary or position is specified. The position of any pointer object with the direct attribute is the next available byte in the space if no position is specified. Space objects are assumed to begin on a 16-byte boundary. You must ensure this condition exists at run-time.

The group of declare statements on the left is equivalent to the group on the right:

```
| DCL SPC X BAS(PTR);                 DCL DD X CHAR(32767) BAS(PTR);
| DCL DD A CHAR(1) DIR;               DCL DD A CHAR(1) DEF(X) POS(1);
| DCL DD B FLT(4) DIR; BDRY(4);       DCL DD B FLT(4) DEF(X) POS(5);
| DCL PTR C DIR; POS(17);             DCL PTR C DEF(X) POS(17);
```

Example: Reset Directive Statement: You can use the reset directive statement to change the name of the space object to be used by subsequent declare statements.

The group of declare statements on the left is equivalent to the group on the right:

```
DCL SPCPTR PTR1;                DCL SPCPTR PTR1;
DCL SPCPTR PTR2;                DCL SPCPTR PTR2;

DCL SPC X BAS(PTR1);            DCL DD X CHAR(32767) BAS(PTR1);
DCL DD A CHAR(2) DIR;           DCL DD A CHAR(2) DEF(X) POS(1);
DCL DD B ZND(5,2) DIR;          DCL DD B ZND(5,2) DEF(X) POS(3);

DCL SPC Y BAS(PTR2);            DCL DD Y CHAR(32767) BAS(PTR2);
DCL DD C CHAR(5) DIR;           DCL DD C CHAR(5) DEF(Y) POS(1);
DCL DD D CHAR(7) DIR;           DCL DD D CHAR(7) DEF(Y) POS(6);

RESET X;
DCL DD E CHAR(3) DIR;           DCL DD E CHAR(3) DEF(X) POS(8);
```

Constants

This section describes the syntax of constant values.

Integer: Integers define signed and unsigned binary scalar data values. The two forms of integers are decimal and hexadecimal. The decimal form is a sequence of digits optionally preceded by a sign. The hexadecimal form is a string of hexadecimal digits delimited with apostrophes and preceded by an H. Neither form may exceed the 4-byte limit on binary numbers. When the value of the integer is between -4095 and +8191, the QPRCRTPG API converts the integer to an immediate operand where it can.

Example:

```
+123
-1
54788

H'0F0D'
H'0123'
H'5E2D1AB4'
```

String: Strings define scalar character string data values. The three types of string constants are character form, hexadecimal form, and Hollerith form.

The character form is a delimited string optionally preceded by a C. Apostrophes or double quotation marks may be used for this form. The hexadecimal form is a delimited string of hexadecimal digits preceded by an X. The Hollerith form is a string of bytes preceded by the count of the number of bytes in the string. The syntax is:

```
< count | string >
```

The count in the preceding syntax is the number of characters in the string. The QPRCRTPG API ensures that the string contains the right number of characters by checking for the > character. No blanks are allowed between < and > unless they are part of the string. The QPRCRTPG API simply flags the constant as in error if the right corner bracket does not appear in the correct position.

Example: The following groups of strings are equivalent:

```
'ABCDE'
C'ABCDE'
X'C1C2C3C4C5'
<5|ABCDE>

'TE''ST'
"TE'ST"
X'E3C57DE2E3'
<5|TE'ST>

'/*'
X'615C'
<2|/*>
```

Packed: Packed constants define packed decimal scalar data values. Packed constants are a string of decimal digits delimited with apostrophes. They can have an embedded decimal point and can be preceded by a sign. P must precede the delimited string. Packed constants have a maximum of 31 significant digits.

Note: You must specify at least one numeric digit.

Example:

```
P'+123.456'
P'1'
P'-1'
P'-123.345345345345'
P'+.00000000000001'
```

Zoned: Zoned constants define zoned decimal scalar data values. The external representation of zoned constants is the same as that for packed constants except that the preceding character is a Z.

Note: You must specify at least one numeric digit.

Example:

```
Z'+123.456'
Z'1'
Z'-1'
Z'-123.345345345345'
Z'+.00000000000001'
```

Floating-Point Constants: Floating-point constants define floating-point scalar data values. You must specify whether the constant is a 4-byte (short floating-point) or an 8-byte (long floating-point) value.

There are two ways to represent floating-point values. First, you can specify floating-point constants as a delimited string of decimal digits possibly with an embedded decimal point and optionally preceded by a sign. An F for short floating-point values or an E for long floating-point values must precede the delimited string. An E in the string determines

the start of the base 10 exponent. You specify the exponent as signed.

Second, you can specify floating-point constants as a string of hexadecimal digits. The delimited string must be preceded by an XF for short floating-point values or an XE for long floating-point values.

Note: You must specify at least one numeric digit.

Example:

Short Floating-Point Values	Long Floating-Point Values
F'0'	E'0'
F'+12'	E'+12'
F'-12.21'	E'-12.21'
F'12.34E2'	E'12.34E2'
F'+3.2345678E-02'	E'+3.2345678E-02'
XF'449A4000'	XE'46CE6F37FFBE8722'
XF'40490FD0'	XE'400921F9F01B866E'

Several special values are allowed:

Short Floating-Point Values	Long Floating-Point Values	
F'MNAN'	E'MNAN'	Masked Not A Number
F'UNAN'	E'UNAN'	Unmasked Not A Number
F'+INF'	E'+INF'	Plus Infinity
F'-INF'	E'-INF'	Minus Infinity

Note: You must use floating-point constants to initialize floating-point data objects.

Name: Names specified in the intermediate representation of a program are a sequence of characters of up to 48 characters in length. You cannot use the following characters as the first character of the name:

```
blank /,;():<+'"%-0123456789
```

You cannot use the following characters in subsequent characters of the name:

```
blank /,;():<+'"%
```

Example:

```
.NAME
NAME
THIS_IS_A_NAME
THIS_IS_A_NAME_2
&NAME
!NAME
?NAME
.0001
```

Note: Symbols that begin with a period (.) are not inserted into the program's symbol table and may not be referred to by the OS/400 debug function.

Comments: Comments, in the intermediate representation of a program, may appear anywhere in the text. Comments are treated as blanks so they are significant in finding tokens. Comments are a string of characters starting with /* and ending with */. If a comment occurs immediately following a semicolon, it prints as a separate line (or a multiple line as required) on the listing. If a comment is embedded in a statement, then it appears as a part of that statement, such as a remark.

Example: The following statements are equivalent:

```
CPYBLA A,B;
CPYBLA A, /* C-> */ B ;
CPYBLA A,B;  /* B is based on C */
```

Blanks: You can use strings of blanks of any length in the intermediate representation of a program. Blanks act as delimiters in finding tokens and in some places are necessary as in separating the opcode and operand in an instruction statement.

Example: The following statements are equivalent:

```
ADDN A,B,C;
ADDN        A    ,     B              , C            ;
```

List ILE Program Information (QBNLPGMI) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified ILE program name	Input	Char(20)
4	Error code	I/O	Char(*)

The List ILE Program Information (QBNLPGMI) API gives information about ILE programs, similar to the Display Program (DSPPGM) command. The information is placed in a user space specified by you.

If an original program model (OPM) program is specified for the qualified ILE program name, an error is returned and the user space is not changed.

You can use the QBNLPGMI API to:

- List modules bound into an ILE program
- List service programs bound to an ILE program
- List copyrights of an ILE program

Authorities and Locks

User Space Authority *CHANGE
User Space Library Authority *USE
User Space Lock *EXCLRD

Program Authority for PGML0100 Format
*USE
Program Authority for other Formats
*READ
Program Library Authority
*READ
Program Lock *SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The user space that is to receive the ILE program 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. The library name can be a specific library name or one of these special values:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The content and format of the information to be returned about the specified programs. One of the following format names may be used:

PGML0100 ILE program module (*MODULE) information. For more information, see "PGML0100 Format."
PGML0200 ILE service program (*SRVPGM) information. For more information, see "PGML0200 Format" on page 52-20.
PGML0500 ILE program copyright (*COPYRIGHT) information. For more information, see "PGML0500 Format" on page 52-20.

Qualified ILE program name
INPUT; CHAR(20)
The name of the ILE program for which the information is to be listed. The first 10 characters contain the ILE program name, and the second 10 characters contain the name of the library where the ILE program is located.

The ILE program name can be a specific ILE program name or one of the following special values:

***ALL** All ILE programs
generic* All ILE programs that begin with this generic prefix. For example, WRK* would include all ILE programs that begin with WRK.

The library name can be a specific library name or one of these special values:

***ALL** All libraries in the system
***ALLUSR** All non-system libraries
***CURLIB** The job's current library
***LIBL** The library list
***USRLIBL** Libraries listed in the user portion of the library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated List

The user space contains:

- 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 descriptions of each field in the list returned, see "Field Descriptions" on page 52-20.

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)	Program name specified
38	26	CHAR(10)	Program 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

PGML0100 Format

The PGML0100 format includes information on all the modules that are bound into the programs specified. The modules will be listed in the user space in the order the modules are bound into the program. You must have a program authority of *USE to use this format. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 52-20.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Program name

Offset			
Dec	Hex	Type	Field
10	A	CHAR(10)	Program library name
20	14	CHAR(10)	Bound module name
30	1E	CHAR(10)	Bound module library name
40	28	CHAR(10)	Source file name
50	32	CHAR(10)	Source file library name
60	3C	CHAR(10)	Source file member name
70	46	CHAR(10)	Module attribute
80	50	CHAR(13)	Module creation date and time
93	5D	CHAR(13)	Source file updated date and time
106	6A	CHAR(10)	Sort sequence table name
116	74	CHAR(10)	Sort sequence table library name
126	7E	CHAR(10)	Language identifier
136	88	BINARY(4)	Optimization level
140	8C	BINARY(4)	Maximum optimization level
144	90	CHAR(10)	Debug data
154	9A	CHAR(6)	Module created on
160	A0	CHAR(6)	Module created for
166	A6	CHAR(20)	Reserved
186	BA	CHAR(1)	User-modified
187	BB	CHAR(13)	Licensed program
200	C8	CHAR(5)	PTF number
205	CD	CHAR(6)	APAR ID
211	D3	CHAR(1)	Reserved
212	D4	BINARY(4)	Module CCSID
216	D8	CHAR(8)	Object control level
224	E0	CHAR(100)	Reserved
324	144	BINARY(4)	Number of SQL statements
328	148	CHAR(18)	Relational database
346	15A	CHAR(10)	Commitment control
356	164	CHAR(10)	Allow copy of data
366	16E	CHAR(10)	Close SQL cursors
376	178	CHAR(10)	Naming convention
386	182	CHAR(10)	Date format
396	18C	CHAR(1)	Date separator
397	18D	CHAR(10)	Time format
407	197	CHAR(1)	Time separator
408	198	CHAR(10)	Delay PREPARE
418	1A2	CHAR(10)	Allow blocking

PGML0200 Format

The PGML0200 format includes information on all the service programs that are bound to the programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Program name
10	A	CHAR(10)	Program library name
20	14	CHAR(10)	Bound service program name
30	1E	CHAR(10)	Bound service program library name
40	28	CHAR(16)	Bound service program signature

PGML0500 Format

The PGML0500 format includes copyright information for the ILE programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Program name
10	A	CHAR(10)	Program library name
20	14	CHAR(4)	Reserved
24	18	BINARY(4)	Copyright length
28	1C	CHAR(256)	Copyright

Field Descriptions

Allow blocking. Whether blocking will be used to improve the performance of certain SQL statements. The possible values are:

**NONE*	Blocking is not used.
**READ*	Blocking is used for read-only cursors when running COMMIT(*NONE) and there are no EXECUTE or EXECUTE IMMEDIATE statements.
**ALLREAD*	Blocking is used for all read-only cursors when running COMMIT(*NONE) or COMMIT(*CHG).
Blank	The module does not contain SQL statements.

Allow copy of data. Whether a copy of the data can be used in the implementation of an SQL query. The possible values are:

**NO*	A copy of the data cannot be used.
**YES*	A copy of the data can be used when needed.

| **OPTIMIZE* The system determines whether a copy of the data is used for optimal performance.
| *Blank* The module does not contain SQL statements.

| **APAR ID**. The module was changed as the result of the authorized program analysis report (APAR) with this identification number. This is blank if the module was not changed at bind time.

| **Bound module library name**. The name of the library containing the module bound into this program at bind time.

| **Bound module name**. The name of the module bound into this program. This is a copy of the module that was bound into this program. It is not the *MODULE object on the system.

| **Bound service program library name**. The name of the library containing the service program bound to the program at bind time. This is the library name in which the activation expects to find the service program at run time. Hexadecimal zeros indicate the library list is used at the time the service program is needed.

| **Bound service program name**. The name of the service program bound to the program.

| **Bound service program signature**. The current signature of the service program at the time it was bound to the program.

| **Close SQL cursors**. Specifies when SQL cursors are implicitly closed and SQL-prepared statements are implicitly discarded. The possible values are:

| **ENDMOD* When the module ends.
| **ENDACTGRP* When the activation group is deleted.
| *Blank* The module does not contain SQL statements.

| **Commitment control**. The level of commitment control that was specified on the SQL precompile. The possible values are:

| **NONE* No commitment control was specified on the SQL precompile. Uncommitted changes in other jobs can be seen.
| **CHG* Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements, and updated, deleted, or inserted rows (records) are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs can be seen.
| **CS* Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements, and updated, deleted, and inserted rows (records) are locked until the end of the unit of work (transaction). A row (record) that is selected but not updated is locked until the next row (record) is selected. Uncommitted changes in other jobs cannot be seen.
| **ALL* Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements, and all rows selected, updated, deleted, and inserted are locked until the end of the unit of work (transaction). Uncommitted changes in other jobs cannot be seen.
| *Blank* The module does not contain SQL statements.

| **Copyright**. The copyright string included in this program.

| **Copyright length**. The length of the copyright string.

| **Date format**. The format used when accessing date-result columns through SQL. All output date fields are returned in this format. For input date strings, the value you specify is used to determine whether the date is a valid format. The values returned are:

| **USA* USA format (mm/dd/yyyy).
| **ISO* International Standards Organization format (yyyy-mm-dd).
| **EUR* European format (dd.mm.yyyy).
| **JIS* Japanese Industrial Standard Christian Era (yyyy-mm-dd).
| **MDY* Month/day/year format (mm/dd/yy).
| **DMY* Day/month/year format (dd/mm/yy).
| **YMD* Year/month/day format (yy/mm/dd).
| **JUL* Julian format (a numeric value from 1 to 365).
| *Blank* The module does not contain SQL statements.

| **Date separator**. The separator used when accessing date-result columns. This information is blank if the module does not contain SQL statements; however, the number of SQL statements field should be checked to determine if the module contains SQL statements. This is because a blank may be specified as a separator value.

| **Debug data**. Whether debug data was generated when this module was created. If debug data exists, the module may be debugged using the source debugger. The possible values are:

| **YES* Debug data was generated.
| **NO* Debug data was not generated.

| **Delay PREPARE**. Whether SQL prepare processing can be delayed until the statement is actually used. The possible values are:

| **YES* Prepare processing can be delayed.
| **NO* Prepare processing cannot be delayed.
| *Blank* The module does not contain SQL statements.

| **Format name specified**. The format used to return the ILE program information to the user space.

| **Language identifier**. Returns the 3-character language identifier used when the module was compiled. The following special values can also be returned:

| **JOBRUN* The language identifier associated with the job at the time the program that the module is bound into runs.

| *Blank* The module does not contain any language identification information.

| **Licensed program**. If the module was part of a licensed program at bind time, this field shows the product number and the level of the licensed program. This is blank if the module is not part of a licensed program at bind time.

| **Maximum optimization level**. The highest level of optimization this module could have at bind time. If observability has been removed from the module, this maximum optimization level value might not be the same as the one specified when the module was created.

| **Module attribute**. The language used in the module. This field can be blank (for example, when a module is created by a compilation process internal to IBM).

| **Module CCSID**. The coded character set identifier (CCSID) for this module.

| **Module created for**. The version, release, and modification level of the operating system for which the module was created. The field has a VvRrMm format, where:

| *Vv* The character V is followed by a 1-character version number.
| *Rr* The character R is followed by a 1-character release level.
| *Mm* The character M is followed by a 1-character modification level.

| **Module created on**. The version, release, and modification level of the operating system on which the module was created. The field has a VvRrMm format, where:

| *Vv* The character V is followed by a 1-character version number.
| *Rr* The character R is followed by a 1-character release level.
| *Mm* The character M is followed by a 1-character modification level.

| **Module creation date and time**. The date and time the module was created. The creation date and time field is in the CYYMMDDHHMMSS format where:

| *C* Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
| *YY* Year
| *MM* Month
| *DD* Day
| *HH* Hour
| *MM* Minute
| *SS* Second

| **Naming convention**. The convention used for naming objects in SQL statements. The possible values are:

| **SQL* The SQL naming convention is used.
| **SYS* The system naming convention is used.
| *Blank* The module does not contain SQL statements.

| **Number of SQL statements**. The number of SQL/400 statements contained in the module. This value is zero if the module does not contain SQL statements.

| **Object control level**. The object control level for the module at the time it was bound into this program. You can compare the object control level of a module to the object control level of a listing to make sure the listing matches the module.

| **Optimization level**. Optimization levels cause the translator to produce machine code that reduces the amount of system resources necessary to run the program. The more optimization, the more efficiently the module runs on the system. Also, with more optimization you may not be able to view variables that have been optimized. The possible values are:

| *30* This level of optimization generates the most efficient code. Variables cannot be changed but can be displayed while the program is being debugged. However, the displayed value of the variable during debugging may not be its actual value.
| *20* Some optimization is performed on the generated code. When the module is being debugged using this level, the variables can be displayed but not changed. This level improves the performance of the module slightly.
| *10* No additional optimization is performed on the generated code. Variables can be displayed and changed when the program is being debugged. With no optimization of the code, this value provides the lowest level of module performance.

| **Program library name**. The name of the library containing the program.

| **Program library name specified**. The program library name that was passed to this API on the call in the qualified ILE program name and library parameter.

| **Program name**. The name of the program.

| **Program name specified**. The program name that was passed to this API on the call in the qualified ILE program name and library parameter.

| **PTF number**. The program temporary fix (PTF) that resulted in the creation of the module. This field is blank for user-created modules.

| **Relational database**. The default relational database that was specified on the SQL precompile. A nonblank value other than *LOCAL specifies the name of the relational database to be resolved through the relational database directory. The following special values can be returned:

| **LOCAL* The module can only access data on the local system.
| *Blank* The module does not contain SQL statements.

| **Reserved**. An ignored field.

| **Sort sequence table name**. The name of the sort sequence table used when the module was compiled. This

| does not apply to SQL statements in the module. The following special values can be returned:

| *HEX | No sort sequence is used.
| *JOBRUN | The sort sequence is the sort sequence value associated with the job at the time the ILE program runs this module.
| *LANGIDSHR | The shared sort sequence for the language identifier is used.
| *LANGIDUNQ | The unique sort sequence for the language identifier is used.

| **Sort sequence table library name**. The name of the library that contained the sort sequence table used when the module was compiled. This does not apply to SQL statements in the module. The following special values can be returned:

| *LIBL | The sort sequence table is found in the library list when the ILE program runs this module.
| *CURLIB | The sort sequence table is found in the current library when the ILE program runs this module.

| **Source file library name**. The name of the library that contains the source file used to create the module. The field is blank if no source file was used to create the module.

| **Source file member name**. The name of the member in the source file. The field is blank if no source file was used to create the module.

| **Source file name**. The name of the source file used to create the module. The field is blank if no source file was used to create the module.

| **Source file updated date and time**. The date and time the member in the source file was last updated. The field is in the same format as the module creation date and time field. The field is blank if no source file was used to create the module.

| **Time format**. The format used when accessing time-result columns through SQL. All output time fields are returned in this format. The values returned are:

| *USA | USA format (hh:mm a.m. or p.m.).
| *ISO | International Standards Organization format (hh.mm.ss).
| *EUR | European format (hh.mm.ss).
| *JIS | Japanese Industrial Standard Christian Era (hh.mm.ss).
| *HMS | Hours/minutes/seconds format (hh:mm:ss).
| Blank | The module does not contain SQL statements.

| **Time separator**. The separator used when accessing time-result columns. This information is blank if the module does not contain SQL statements; however, the number of SQL statements field should be checked to determine if the module contains SQL statements. This is because a blank may be specified as a separator value.

| **User-modified**. Whether the module was changed by the user at bind time. The possible values are:

| 0 | The user did not change the module.
| 1 | The user changed the module.

| **User space library name specified**. The user space library name that was passed to this API on the call in the qualified user space name parameter.

| **User space library name used**. The name of the library that contains the user space that receives the ILE program information requested.

| **User space name specified**. The user space name that was passed to this API on the call in the qualified user space name parameter.

| **User space name used**. The name of the user space that receives the ILE program information requested.

| Error Messages

| CPF24B4 E Severe error while addressing parameter list.
| CPF3C20 E Error found by program &1.
| CPF3C21 E Format name &1 is not valid.
| CPF3CF1 E Error code parameter not valid.
| CPF5CF5 E &1 in library &2 not ILE program.
| CPF5CF6 E Program &1 not valid special value.
| CPF811A E User space &4 in &9 damaged.
| CPF9570 E Error occurred creating or accessing debug data.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9804 E Object &2 in library &3 damaged.
| CPF9806 E Cannot perform function for object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9811 E Program &1 in library &2 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9821 E Not authorized to program &1 in library &2.
| CPF9830 E Cannot assign library &1.
| CPF9838 E User profile storage limit exceeded.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

| List Service Program Information (QBNLSPGM) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified service program name	Input	Char(20)
4	Error code	I/O	Char(*)

The List Service Program Information (QBNLSPGM) API gives information about service programs, similar to the Display Service Program (DSPSRVPGM) command. The information is placed in a user space specified by you.

You can use the QBNLSPGM API to:

- List modules bound into a service program
- List service programs bound to a service program
- List copyrights of a service program
- List procedure export information of a service program
- List data export information of a service program
- List signatures of a service program

Authorities and Locks

User Space Authority *CHANGE
User Space Library Authority
*USE
User Space Lock *EXCLRD
Service Program Authority for SPGL0100 Format
*USE
Service Program Authority for other Formats
*READ
Service Program Library Authority
*READ
Service Program Lock *SHRRD

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The user space that is to receive the service program information. 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 library name can be a specific library name or one of these special values:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The content and format of the information to be returned about the specified service program(s). One of the following format names may be used:

SPGL0100 Service program module (*MODULE) information. For more information, see "SPGL0100 Format" on page 52-25.

SPGL0200 Service program (*SRVPGM) information. For more information, see "SPGL0200 Format" on page 52-25.

SPGL0500 Service program copyright (*COPYRIGHT) information. For more information, see "SPGL0500 Format" on page 52-26.

SPGL0600 Service program procedure export (*PROCEXP) information. For more information, see "SPGL0600 Format" on page 52-26.

SPGL0700 Service program data export (*DTAEXP) information. For more information, see "SPGL0700 Format" on page 52-26.

SPGL0800 Service program signature (*SIGNATURE) information. For more information, see "SPGL0800 Format" on page 52-26.

Qualified service program name
INPUT; CHAR(20)
The name of the service program for which the information is to be listed. The first 10 characters contain the service program name. The second 10 characters contain the name of the library where the service program is located.

The service program name can be a specific service program name or one of the following special values:

***ALL** All service programs
generic* All service programs that begin with this generic prefix. For example, WRK* lists information for all service programs that begin with WRK to which you are authorized.

The library name can be a specific library name or one of these special values:

***ALL** All libraries in the system
***ALLUSR** All non-system libraries
***CURLIB** The job's current library
***LIBL** The library list
***USRLIBL** Libraries listed in the user portion of the library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated List

The user space contains:

- 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 descriptions of each field in the list returned, see "Field Descriptions" on page 52-26.

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)	Service program name specified
38	26	CHAR(10)	Service program library name specified

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name used
10	A	CHAR(10)	User space library name used

SPGL0100 Format

The SPGL0100 format includes information on all the modules that are bound into the programs specified. The modules are listed in the user space in the order the modules are bound into the program. You must have a service program authority of *USE to use this format. The following table shows how this information for each module is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 52-26.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	CHAR(10)	Bound module name
30	1E	CHAR(10)	Bound module library name
40	28	CHAR(10)	Source file name
50	32	CHAR(10)	Source file library name
60	3C	CHAR(10)	Source file member name
70	46	CHAR(10)	Module attribute
80	50	CHAR(13)	Module creation date and time
93	5D	CHAR(13)	Source file updated date and time
106	6A	CHAR(10)	Sort sequence table name
116	74	CHAR(10)	Sort sequence table library name
126	7E	CHAR(10)	Language identifier
136	88	BINARY(4)	Optimization level
140	8C	BINARY(4)	Maximum optimization level
144	90	CHAR(10)	Debug data
154	9A	CHAR(6)	Module created on
160	A0	CHAR(6)	Module created for
166	A6	CHAR(20)	Reserved
186	BA	CHAR(1)	User-modified
187	BB	CHAR(13)	Licensed program
200	C8	CHAR(5)	PTF number
205	CD	CHAR(6)	APAR ID
211	D3	CHAR(1)	Reserved
212	D4	BINARY(4)	Module CCSID
216	D8	CHAR(8)	Object control level
224	E0	CHAR(100)	Reserved
324	144	BINARY(4)	Number of SQL statements
328	148	CHAR(18)	Relational database
346	15A	CHAR(10)	Commitment control
356	164	CHAR(10)	Allow copy of data
366	16E	CHAR(10)	Close SQL cursor
376	178	CHAR(10)	Naming convention
386	182	CHAR(10)	Date format
396	18C	CHAR(1)	Date separator
397	18D	CHAR(10)	Time format
407	197	CHAR(1)	Time separator
408	198	CHAR(10)	Delay PREPARE
418	1A2	CHAR(10)	Allow blocking

SPGL0200 Format

The SPGL0200 format includes information on all the service programs that are bound to the programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 52-26.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	CHAR(10)	Bound service program name

Offset		Type	Field
Dec	Hex		
30	1E	CHAR(10)	Bound service program library name
40	28	CHAR(16)	Bound service program signature

SPGL0500 Format

The SPGL0500 format includes copyright information for the service programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	CHAR(4)	Reserved
24	18	BINARY(4)	Copyright length
28	1C	CHAR(256)	Copyright

SPGL0600 Format

The SPGL0600 format includes procedure export information for the service programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	BINARY(4)	Procedure export CCSID
24	18	BINARY(4)	Procedure export name length
28	1C	CHAR(256)	Procedure export name

SPGL0700 Format

The SPGL0700 format includes data export information for the service programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	BINARY(4)	Data item CCSID
24	18	BINARY(4)	Data item name length
28	1C	CHAR(256)	Data item name

SPGL0800 Format

The SPGL0800 format includes signature information for the service programs specified. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Service program name
10	A	CHAR(10)	Service program library name
20	14	CHAR(16)	Signature

Field Descriptions

Allow blocking. Whether blocking is used to improve the performance of certain SQL statements. The possible values are:

**NONE*	Blocking is not used.
**READ*	Blocking is used for read-only data cursors when running COMMIT(*NONE) and there are no EXECUTE or EXECUTE IMMEDIATE statements.
**ALLREAD*	Blocking is used for all read-only cursors when running COMMIT(*NONE) or COMMIT(*CHG).
Blank	The module does not contain SQL statements.

Allow copy of data. Whether a copy of the data can be used in the implementation of an SQL query. The possible values are:

**NO*	A copy of the data cannot be used.
**YES*	A copy of the data can be used when needed.
**OPTIMIZE*	The system determines whether a copy of the data is used for optimal performance.
Blank	The module does not contain SQL statements.

APAR ID. The module was changed as the result of the authorized program analysis report (APAR) with this identification number. This is blank if the module was not changed at bind time.

Bound module library name. The name of the library containing the module bound into this service program at bind time.

Bound module name. The name of the module bound into this service program. This is a copy of the module that was bound into this service program. It is not the *MODULE object on the system.

Bound service program library name. The name of the library containing the service program bound to this service program at bind time. This is the library name in which the activation expects to find the service program at run time. Hexadecimal zeros indicate the library list is used at the time the service program is needed.

Bound service program name. The name of the service program bound to this service program.

Bound service program signature. The current signature of the service program at the time the service program was bound to this service program.

Close SQL cursor. Specifies when SQL cursors are implicitly closed and SQL-prepared statements are implicitly discarded. The possible values are:

*ENDMOD	When the module ends.
*ENDACTGRP	When the activation group is deleted.
Blank	The module does not contain SQL statements.

Commitment control. The level of commitment control that was specified on the SQL precompile. The possible values are:

*NONE	No commitment control was specified on the SQL precompile. Uncommitted changes in other jobs can be seen.
*CHG	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). Updated, deleted, and inserted rows (records) are locked until the end of the unit of work. Uncommitted changes in other jobs can be seen.
*CS	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). Updated, deleted, and inserted rows (records) are locked until the end of the unit of work. A row (record) that is selected but not updated is locked until the next row (record) is selected. Uncommitted changes in other jobs cannot be seen.
*ALL	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). All rows selected, updated, deleted, and inserted are locked until the end of the unit of work. Uncommitted changes in other jobs cannot be seen.
Blank	The module does not contain SQL statements.

Copyright. The copyright string included in this service program.

Copyright length. The length of the copyright string.

Data item name. Service program data items that are allowed to be exported.

Data item CCSID. The coded character set identifier (CCSID) of this data item.

Data item name length. The length of the data item name.

Date format. The format used when accessing date-result columns through SQL. All output date fields are returned in this format. For input date strings, the value you specify is used to determine whether the date is a valid format. The values returned are:

*USA	USA format (mm/dd/yyyy).
*ISO	International Standards Organization format (yyyy-mm-dd).
*EUR	European format (dd.mm.yyyy).
*JIS	Japanese Industrial Standard Christian Era (yyyy-mm-dd).
*MDY	Month/day/year format (mm/dd/yy).
*DMY	Day/month/year format (dd/mm/yy).
*YMD	Year/month/day format (yy/mm/dd).
*JUL	Julian format (a numeric value from 1 to 365).
Blank	The module does not contain SQL statements.

Date separator. The separator used when accessing date-result columns. This information is blank if the module does not contain SQL statements. However, the number of SQL statements field should be checked to determine if the module contains SQL statements. This is because a blank may be specified as a separator value.

Debug data. Whether debug data was generated when this module was created. If debug data exists, the module may be debugged using the source debugger. The possible values are:

*YES	Debug data was generated.
*NO	Debug data was not generated.

Delay PREPARE. Whether SQL prepare processing can be delayed until the statement is actually used. The possible values are:

*YES	Prepare processing can be delayed.
*NO	Prepare processing cannot be delayed.
Blank	The module does not contain SQL statements.

Format name specified. The format name that was passed to this API on the call in the format parameter.

Language identifier. Returns the 3-character language identifier used when the module was compiled. The following special values can also be returned:

| **JOBRUN* The language identifier associated with the job at the time the service program into which the module is bound runs.
| *Blank* The module does not contain any language identification information.

| **Licensed program**. If the module was part of a licensed program at bind time, this field shows the product number and the level of the licensed program. This is blank if the module is not part of a licensed program at bind time.

| **Maximum optimization level**. The highest level of optimization this module could have at bind time. If observability has been removed from the module, this maximum optimization level value might not be the same as the one specified at module creation.

| **Module attribute**. The language in which the module is written. This field can be blank (for example, a module created by a compilation process internal to IBM).

| **Module CCSID**. The coded character set identifier (CCSID) for this module.

| **Module created for**. The version, release, and modification level of the operating system for which the module was created. The field has a VvRrMm format, where:

| *Vv* The character V is followed by a 1-character version number.
| *Rr* The character R is followed by a 1-character release level.
| *Mm* The character M is followed by a 1-character modification level.

| **Module created on**. The version, release, and modification level of the operating system on which the module was created. The field has a VvRrMm format, where:

| *Vv* The character V is followed by a 1-character version number.
| *Rr* The character R is followed by a 1-character release level.
| *Mm* The character M is followed by a 1-character modification level.

| **Module creation date and time**. The date and time the module was created. The creation date and time field is in the CYYMMDDHHMMSS format, where:

| *C* Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
| *YY* Year
| *MM* Month
| *DD* Day
| *HH* Hour
| *MM* Minute
| *SS* Second

| **Naming convention**. The convention used for naming objects in SQL statements. The possible values are:

| **SQL* The SQL naming convention is used.
| **SYS* The system naming convention is used.
| *Blank* The module does not contain SQL statements.

| **Number of SQL statements**. The number of SQL/400 statements contained in the module. This value is zero if the module does not contain SQL statements.

| **Object control level**. The object control level for the module at the time it was bound into this service program. You can compare the object control level of a module to the object control level of a listing to make sure they match.

| **Optimization level**. Optimization levels cause the translator to produce machine code that reduces the amount of system resources necessary to run the program. The more optimization, the more efficiently the module runs on the system. Also, with more optimization you may not be able to view variables that have been optimized. The possible values are:

| *30* This level of optimization generates the most efficient code. Variables cannot be changed but can be displayed while the program is being debugged. However, the displayed value of the variable during debugging may not be its actual value.
| *20* Some optimization is performed on the generated code. When the module is being debugged using this level, the variables can be displayed but not changed. This level improves the performance of the module slightly.
| *10* No additional optimization is performed on the generated code. Variables can be displayed and changed when the program is being debugged. With no optimization of the code, this value provides the lowest level of module performance.

| **Procedure export name**. Service program procedures that are allowed to be exported.

| **Procedure export CCSID**. The coded character set identifier (CCSID) of this procedure export.

| **Procedure export name length**. The length of the procedure export name.

| **PTF number**. The program temporary fix (PTF) that resulted in the creation of the module. This field is blank for user-created modules.

| **Relational database**. The default relational database that was specified on the SQL precompile. A nonblank value other than *LOCAL specifies the name of the relational database to be resolved through the relational database directory. The following special values can be returned:

| **LOCAL* The module can only access data on the local system.
| *Blank* The module does not contain SQL statements.

| **Reserved**. An ignored field.

| **Service program library name**. The name of the library containing the service program.

Service program library name specified. The service program library name that was passed to this API on the call in the qualified service program name parameter.

Service program name. The name of the service program.

Service program name specified. The service program name that was passed to this API on the call in the qualified service program name parameter.

Signature. A valid signature of this service program.

Sort sequence table name. The name of the sort sequence table used when the module was compiled. This does not apply to SQL statements in the module. The following special values can be returned:

**HEX*	No sort sequence is used.
**JOBRUN*	The sort sequence value associated with the job at the time the service program runs this module is used.
**LANGIDSHR*	The shared sort sequence for the language identifier is used.
**LANGIDUNQ*	The unique sort sequence for the language identifier is used.

Sort sequence table library name. The name of the library that contained the sort sequence table used when the module was compiled. This does not apply to SQL statements in the module. The following special values can be returned:

**LIBL*	The sort sequence table is found in the library list when the service program runs this module.
**CURLIB*	The sort sequence table is found in the current library when the service program runs this module.

Source file library name. The name of the library that contains the source file used to create the module. The field is blank if no source file was used to create the module.

Source file member name. The name of the member in the source file. The field is blank if no source file was used to create the module.

Source file name. The name of the source file used to create the module. The field is blank if no source file was used to create the module.

Source file updated date and time. The date and time the member in the source file was last updated. The field is in the same format as the module creation date and time field. The field is blank if no source file was used to create the module.

Time format. The format used when accessing time-result columns through SQL. All output time fields are returned in this format. The values returned are:

**USA*	USA format (hh:mm a.m. or p.m.).
**ISO*	International Standards Organization format (hh.mm.ss).
**EUR*	European format (hh.mm.ss).
**JIS*	Japanese Industrial Standard Christian Era (hh.mm.ss).
**HMS*	Hours/minutes/seconds format (hh:mm:ss).
Blank	The module does not contain SQL statements.

Time separator. The separator used when accessing time-result columns. This information is blank if the module does not contain SQL statements. However, the number of SQL statements field should be checked to determine if the module contains SQL statements. This is because a blank may be specified as a separator value.

User-modified. Whether the module was changed by the user. The possible values are:

0	The user did not change the module.
1	The user changed the module.

User space library name specified. The user space library name that was passed to this API on the call in the qualified user space name parameter.

User space library name used. The name of the library that contains the user space that receives the service program information requested.

User space name specified. The user space name that was passed to this API on the call in the qualified user space name parameter.

User space name used. The name of the user space that receives the service program information requested.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3C20 E Error found by program &1.
CPF3C21 E Format name &1 is not valid.
CPF3CF1 E Error code parameter not valid.
CPF5CF5 E &1 in library &2 not ILE program.
CPF5CF6 E Program &1 not valid special value.
CPF811A E User space &4 in &9 damaged.
CPF9570 E Error occurred creating or accessing debug data.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9804 E Object &2 in library &3 damaged.
CPF9806 E Cannot perform function for object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
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.

Process Commands (QCAPCMD) API

Parameters

Required Parameter Group:

1	Source command string	Input	Char(*)
2	Length of source command string	Input	Binary(4)
3	Options control block	Input	Char(*)
4	Options control block length	Input	Binary(4)
5	Options control block format	Input	Char(8)
6	Changed command string	Output	Char(*)
7	Length available for changed command string	Input	Binary(4)
8	Length of changed command string available to return	Output	Binary(4)
9	Error code	I/O	Char(*)

The Process Commands (QCAPCMD) API is used to perform command analyzer processing on command strings. You can check or run CL commands from HLLs as well as check syntax for specific source definition types.

You can use the QCAPCMD API to:

- Check the syntax of a command string prior to running it
- Prompt the command and receive the changed command string
- Run a command from an HLL

Authorities and Locks

Any command *USE

Required Parameter Group

Source command string
INPUT; CHAR(*)
The command string to be prompted for or run.

Length of source command string
INPUT; BINARY(4)
The length of the source command string. Valid values are between 1 and 6000. Message CPF3CFD will result for values outside this range. This length can include trailing blanks.

Options control block
INPUT; CHAR(*)
The options that control the handling of the command string. The layout of this parameter is the CPOP0100 format. For more information, see "CPOP0100 Format."

Options control block length
INPUT; BINARY(4)
The length of the options control block. A minimum length of 20 is required for the CPOP0100 format.

Options control block format
INPUT; CHAR(8)
The format of the options control block. CPOP0100 is the only valid value. For more information, see "CPOP0100 Format."

Changed command string
OUTPUT; CHAR(*)
The rebuilt command string. This is the updated command string, which includes changes from prompting, ordering of parameters, and the addition of keywords. This string may be substantially longer than the source command string. If an error occurs that prevents the command string from being rebuilt, this field is not changed. No padding is performed on the value returned. The length of changed command string available to return parameter should be used to determine how much data is returned.

Length available for changed command string
INPUT; BINARY(4)
The length available to return the updated command string. If an updated command string is not desired, a special value of 0 may be specified. This value must be a positive number or zero.

Length of changed command string available to return
OUTPUT; BINARY(4)
The length of the changed command string returned or available to return. If zero is specified for the length available for changed command string parameter, this value is not set. If an error occurs that prevents the command string from being rebuilt, this field is zero. If the changed command string parameter is not large enough to hold the entire rebuilt command string, this value is the total length available. The changed command string is truncated.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

CPOP0100 Format

The CPOP0100 format includes information on the contents of the options control block parameter. The following table shows how this information is organized. For detailed descriptions of the fields in the list, see "Field Descriptions" on page 52-31.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Type of command processing
4	4	CHAR(1)	DBCS data handling
5	5	CHAR(1)	Prompter action
6	6	CHAR(1)	Command string syntax

Offset		Type	Field
Dec	Hex		
7	7	CHAR(13)	Reserved (Must be set to hexadecimal zeros)

Field Descriptions

Command string syntax. Whether command processing should be done in AS/400 mode or System/38 mode. The possible values are:

0 Use AS/400 syntax. The specification of qualified objects is in the format library/object.

1 Use System/38 syntax. The specification of qualified objects is in the format object.library. The system searches the QUSER38 library (if it exists) and the QSYS38 library for the command even though these libraries are not in the library list.

DBCS data handling. Whether the command analyzer should handle the SO/SI characters as DBCS delimiters. It is valid with all classes of command processing. This option is the equivalent of specifying IGC as the command processing mode pointer parameter on the Execute Command (QCMDEXC) and Check Command Syntax (QCMDCHK) API. The possible values are:

0 Ignore DBCS data.

1 Handle DBCS data.

Prompter action. Indicates whether the prompter should be called on a command string.

0 Never prompt the command. This will prevent a command prompt even if selective prompting characters are present in the command string.

1 Always prompt the command. This forces a command prompt even if selective prompting characters are not present in the command string.

2 Prompt the command if selective prompting characters are present in the command string. A CPF3CF1 exception is sent if this value is specified with types of command processing values 4 through 8.

Reserved. This area must be set to hexadecimal zeros.

Type of command processing. The type of command processing to be performed by the system. The following processes can occur:

0 Command running. The processing for this type is the same as that performed by the QCMDEXC API. Commands processed must have a value of *EXEC on the ALLOW parameter of the Create Command (CRTCMD) or the Change Command (CHGCMD) command.

1 Command syntax check. The processing for this type is the same as that performed by the QCMDCHK API.

2 Command line running. This processing is like that provided by the QCMDEXC API but with the following additions:

- Limited user checking is performed.
- Prompting for missing required parameters is performed.
- If the System/36 environment is active and the commands are System/36 commands, the System/36 environment runs the commands.

This type processes commands with entry codes of `Job: I` (value of *INTERACT on the ALLOW parameter of the CRTCMD or CHGCMD command). While this type is meant to implement an interactive command line, it can be used in batch. When used in a batch job, the entry code for the command must be `Job: B`. Limited user checking and System/36 environment processing is done while prompting options are ignored.

3 Command line syntax check. This processing provides the check only complement of type 2 (command line running). The check option performs all checks against CL rules. The System/36 environment is not called.

4 CL program statement. The command string is checked according to the rules for CL programs (source entry utility (SEU) member type of CLP). Commands may not be run with this type. Command prompts include a prompt for a command label and comment. Variable names are allowed. Commands processed for this type must be defined with entry codes of `Pgm: B`, `Pgm: I`, or `Pgm: B,I`. They have values of *BPGM or *IPGM on the ALLOW parameter of the CRTCMD or CHGCMD command.

5 CL input stream. The command string is checked according to the rules for CL batch jobs (SEU member type of CL). Commands may not be run. Command prompts include a prompt for comment. Variable names are not allowed.

6 Command definition statements. The command string is checked according to the rules for command definition (SEU member type of CMD). Commands may not be run. The commands are restricted to CMD, PARM, ELEM, QUAL, DEP, and PMTCTL.

7 Binder definition statements. The command string is checked according to the rules for binder definition (SEU member type of BND). Commands may not be run. The commands are restricted to STRPGMEXP, ENDPGMEXP, and EXPORT.

8 User-defined option. This option allows a user to create user-defined option command strings similar to those used by the programming development manager (PDM). It allows checking and creating a command string for future use with types 0 through 3 except that variables are allowed. The command string produced may not be directly operable. That is, if CL variables were specified in the command string, the user must perform a substitution prior to using the API.

Usage Considerations

The prompt actions controlled by the prompter option field in the option control block have the following considerations.

- For commands with a value of 0, 1, 2, or 3 for the type of command processing field, a prompt occurs when 2 is specified for the prompter option field if:
  - Selective prompting is specified in the source command string parameter.
  - The job is running interactively.
- If this API is called in a batch job with a valid prompt request, it is ignored. A valid prompt request is issued by specifying
  - 1 for the prompter option field
  - 2 for the prompter option field with selective prompting characters in the command string.

Calls of the API in batch jobs with values of 4, 5, 6, or 7 for the type of command processing field are processed. However, prompting requests are ignored.

Error Messages

CPF0008 E Value in option control block not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF3C1D E Length specified in parameter &1 not valid.
CPF3C20 E Error found by program &1.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF9872 E Program &1 in library &2 ended. Reason code &3.
xxxnnnn E Any escape message issued by any command may be returned. The messages listed previously are those issued by this API. Once the API has called the command analyzer, any message issued as an escape may appear.

Retrieve Command Information (QCDRCMDI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified command name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Command Information (QCDRCMDI) API retrieves information from a command definition object and places it into a single variable in the calling program. The amount of information returned depends on the size of the variable. The information returned is the same information returned by the Display Command (DSPCMD) command.

You can use the QCDRCMDI API to retrieve any operable command. This includes both interactive (such as Display Program (DSPPGM) and Create Library (CRTLIB)) and non-interactive (such as DO, IF, and ELSE) commands. It does not include command definition statements that appear in command source, such as CMD, DEP, ELEM, PARM, PARMCTL, and QUAL.

Authorities and Locks

Command Definition Object Authority *USE
Library Authority *USE
Command Definition Object Lock *SHRRD

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. The minimum size for this area is 8 bytes. You can specify the size of this area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If this value is larger than the actual size of the receiver variable, the result may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the command information to be returned. One of the following format names may be used:

CMDI0100 Basic command information. For more information, see "CMDI0100 Format" on page 52-33.
CMDI0200 Complete command information. For more information, see "CMDI0200 Format" on page 52-33.

Qualified command name
INPUT; CHAR(20)
The name of the command whose values are to be retrieved. The first 10 characters contain the name of the command. The second 10 characters contain the name of the library where the command is located.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

CMDI0100 Format

The following table describes the information that is returned in the receiver variable for the CMDI0100 format. For detailed descriptions of the fields, see "Field Descriptions" on page 52-34.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Command name
18	12	CHAR(10)	Command library name
28	1C	CHAR(10)	Command processing program name
38	26	CHAR(10)	Command processing program library name
48	30	CHAR(10)	Source file name
58	3A	CHAR(10)	Source file library name
68	44	CHAR(10)	Source file member name
78	4E	CHAR(10)	Validity check program name
88	58	CHAR(10)	Validity check program library name
98	62	CHAR(10)	Mode information
108	6C	CHAR(15)	Where allowed to run
123	7B	CHAR(1)	Allow limited user
124	7C	BINARY(4)	Maximum positional parameters
128	80	CHAR(10)	Prompt message file name
138	8A	CHAR(10)	Prompt message file library name
148	94	CHAR(10)	Message file name
158	9E	CHAR(10)	Message file library name
168	A8	CHAR(10)	Help panel group name
178	B2	CHAR(10)	Help panel group library name
188	BC	CHAR(10)	Help identifier
198	C6	CHAR(10)	Search index name
208	D0	CHAR(10)	Search index library name
218	DA	CHAR(10)	Current library
228	E4	CHAR(10)	Product library
238	EE	CHAR(10)	Prompt override program name
248	F8	CHAR(10)	Prompt override program library name
258	102	CHAR(6)	Restricted to target release
264	108	CHAR(50)	Text description
314	13A	CHAR(2)	Command processing program call state
316	13C	CHAR(2)	Validity checker program call state
318	13E	CHAR(2)	Prompt override program call state
320	140	CHAR(30)	Reserved

CMDI0200 Format

The following table describes the information that is returned in the receiver variable for the CMDI0200 format. For detailed descriptions of the fields, see "Field Descriptions" on page 52-34.

Offset			
Dec	Hex	Type	Field
0	0		Returns everything from format CMDI0100
350	15E	CHAR(10)	REXX source file name
360	168	CHAR(10)	REXX source file library name
370	172	CHAR(10)	REXX source file member name
380	17C	CHAR(10)	REXX command environment name
390	186	CHAR(10)	REXX command environment library name
400	190	CHAR(40)	Reserved
440	1B8	BINARY(4)	Number of REXX exit entries
444	1BC	BINARY(4)	Length of a REXX exit entry
Format of a REXX exit entry (repeated by the number of REXX exit entries)			
See note	See note	CHAR(10)	REXX exit program name
See note	See note	CHAR(10)	REXX exit program library name
See note	See note	BINARY(4)	REXX exit code

Note: The decimal and hexadecimal offsets to the above 3 fields depend on the number of REXX exit entries and the length of a REXX exit entry. The REXX exit entry fields (currently REXX exit program name, REXX exit program library name, and REXX exit code) repeat, in the order listed, by the number of REXX exit entries defined for this command.

Field Descriptions

For more information on the following fields, refer to the documentation for the Create Command (CRTCMD) command in the *CL Reference*.

Allow limited user. Whether or not a user with limited authorities is allowed to run this command. The possible values are 0 (*NO) or 1 (*YES).

Bytes available. The length of all data available for the requested format. All available data is returned if enough space is provided.

Bytes returned. The length of all data actually returned. If the data is truncated because the receiver variable is not large enough to hold the data, this value is less than the bytes available.

Command library name. The name of the library in which the command description resides.

Command name. The name of the command description about which information is being returned.

Command processing program call state. The state the command processing program will run in.

Command processing program library name. The name of the library in which the process command program resides. This field is blank if the command processing program name contains the special value *REXX.

Command processing program name. The name of the program that accepts parameters from the command and processes the command. The possible values are:

**REXX*	The REXX fields returned in the CMDI0200 format contain valid information about the command.
CPP-program-name	The command processing program name.

Current library. The name of the library used as the current library during the processing of this command. The possible values are:

**NOCHG*	The current library does not change for the processing of this command. If the current library is changed during processing of the command, the change remains in effect after command processing is complete.
**CRTDFT*	No current library is active during processing of the command. The current library that was active before command processing began is restored when processing is completed.
library-name	The name of the library that is used as the current library. When command processing is completed, the current library is restored to its previous value.

Help identifier. The name of the general help module for the names of the help identifiers for this command. The possible values are:

**NONE*	No help is available.
**CMD*	The name of the command is used.
help-ID-name	A user-specified help module was used.

Help panel group library name. The name of the library in which the panel group resides.

Help panel group name. The name of the panel group in which the online help information exists for this command. If *NONE is returned, no help is available for this command.

Length of a REXX exit entry. The length of one REXX exit entry. This value is currently 24. There are 10 bytes for the REXX exit program name, 10 bytes for the REXX exit library name, and 4 bytes for the REXX exit code.

Maximum positional parameters. The maximum number of parameters than can be coded in a positional manner for this command. The possible values are:

-1	No maximum positional coding limit was specified for this command.
0 through 75	The maximum number of parameters that can be coded in a positional manner for this command.

Message file library name. The name of the library in which the message file resides.

Message file name. The message file from which messages identified on the DEP statements used to define the command are retrieved.

Mode information. The mode of operating environment to which the command applies. The characters of this field are as follows, and they can have a value of 0 (does not apply) or 1 (does apply):

1	Production mode
2	Debug mode
3	Service mode
4-10	Reserved

Number of REXX exit entries. The number of times the REXX exit entries are repeated. These fields are REXX exit program name, REXX exit program library name, and REXX exit code.

Product library. The name of the product library that is in effect during the processing of the command. The possible values are:

**NOCHG*	The product library does not change for the processing of this command.
**NONE*	There is no product library in the job's library list.
library-name	The name of the library that is used as the product library during the processing of the command.

Prompt message file library name. The name of the library in which the prompt message file resides.

Prompt message file name. The name of the message file that contains the prompt text for this command. If *NONE is returned, no message file was specified for prompt text.

Prompt override program call state. The state the prompt override program will run in.

Prompt override program library name. The name of the library in which the prompt override program resides.

Prompt override program name. This is the name of the prompt override program that replaces default values (on the prompt display) with the current actual values for the parameter. If *NONE is returned, no prompt override program was specified for this command.

Reserved. An ignored field.

Restricted to target release. The version, release, and modification level to which this command is restricted. If this field is blank, the command can be used in the current release. This applies only to a command used in a CL program. It must match the contents of the target release parameter on the Create CL Program (CRTCLPGM) command. See the CRTCLPGM command for more information. This field has the format VvRrMm, where:

Vv The character V is followed by a 1-character version number.
Rr The character R is followed by a 1-character release level.
Mm The character M is followed by a 1-character modification level.

REXX command environment library name. The name of the library in which the REXX command environment program resides.

REXX command environment name. The command environment program that is active when the REXX CPP starts to run. The REXX interpreter calls this program to process commands encountered in the REXX procedure. The possible values are:

**COMMAND*	The AS/400 system control language command environment is used.
**CPICOMM*	The Common Programming Interface (CPI) for Communications command environment is used. CPICOMM is the command environment used for CL commands that are embedded within a REXX procedure.
program-name	The name of the program to process commands found in the REXX procedure.

REXX exit code. A value which controls the conditions in which the REXX exit program is called. The possible values are:

2 The exit program is called whenever an external function or subroutine has been called by the REXX program. The exit program is responsible for locating and calling the requested routine.
3 The exit program is called whenever the interpreter is going to call a command. The exit program is responsible for locating and calling the command.
4 The exit program is called whenever a REXX instruction or function attempts an operation on the REXX external data queue.
5 The exit program is called when session input or output operations are attempted.
7 The exit program is called after running each clause of the REXX procedure to determine whether it must be stopped.
8 The exit program is called after running each clause of the REXX program to check if tracing must be turned on or off.
9 The exit program is called before interpretation of the first instruction of a REXX procedure.
10 The exit program is called after interpretation of the last instruction of a REXX procedure.

REXX exit program library name. The name of the library in which the REXX exit program resides.

REXX exit program name. The exit program used when the REXX interpreter is started under the conditions specified by the REXX exit code for this program.

REXX source file library name. The name of the library in which the REXX source file resides.

REXX source file member name. The name of the source file member that contains the REXX procedure that is the command processing program.

REXX source file name. The name of the source file that contains the REXX procedure that is the command processing program. The possible values are:

QREXSRC	The IBM-supplied source file, QREXSRC, contains the source member that is used.
source-file-name	The name of the REXX source file that is used.

Search index library name. The name of the library in which the help search index resides.

Search index name. The help search index to be used when the search index function key is pressed. The possible values are:

**SYSTEM*	The system index file, QHSS1.
**NONE*	No help search index is used.
search-index-name	The name of the help search index that is used.

Source file library name. The name of the library in which the source file resides.

Source file member name. The name of the source file member that contains the command definition statements used to create the command.

Source file name. The name of the source file that contains the source file member used to create the command.

Text description. The user text, if any, used to briefly describe the command and its function.

Validity check program call state. The state the validity check program will run in.

Validity check program library name. The name of the library in which the validity checking program resides.

Validity check program name. The name of a program that performs additional user-defined validity checking on the parameters in the command. If *NONE is returned, no separate user-defined validity checking is done for this command. All validity checking is done by the command analyzer and the command processing program.

Where allowed to run. The environments in which this command is allowed to run. The characters of this field are as follows, and they can have a value of 0 (does not apply) or 1 (does apply):

1 Batch program (*BPGM)
2 Interactive program (*IPGM)
3 Can be run using QCMDEXC, QCAEXEC, or QCAPCMD (*EXEC)
4 Interactive job (*INTERACT)
5 Batch job (*BATCH)
6 Batch REXX procedure (*BREXX)
7 Interactive REXX procedure (*IREXX)
8–15 Reserved

Error Messages

CPF2150 E Object information function failed.
CPF2151 E Operation failed for &2 in &1 type *&3.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF6250 E Cannot display or retrieve command &1 in library &2.
CPF8103 Command &4 in &9 damaged.
CPF8122 E &8 damage on library &4.
CPF8123 E Damage on object information for library &4.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &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.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Program Associated Space (QCLRPGAS) API

Parameters

Required Parameter Group:

1	Output data buffer	Output	Char(*)
2	Length of output data buffer	Input	Binary(4)
3	Qualified program name	Input	Char(20)
4	Call stack counter	Input	Binary(4)
5	Data handle	Input	Char(16)
6	Length of data available	Output	Binary(4)
7	Error code	I/O	Char(*)

The Retrieve Program Associated Space (QCLRPGAS) API allows you to retrieve information from the associated space of an original program model (OPM), user-state program in the user domain. This API is intended to be used only on programs created by the Create Program (QPRCRTPG) API.

This API is used to retrieve information that was previously stored in the associated space. For example, it can be used by compilers at run time to retrieve information about the compilation process that was previously stored. The associated space of a program object is not a replacement for data areas or user spaces. It is recommended for static data only.

Authorities and Locks

Program Library Authority *USE
Program Authority *ALL
Program Lock *EXCLRD

Required Parameter Group

Output data buffer
OUTPUT; CHAR(*)
The information retrieved from the associated space. This information contains scalar data only. If information that was previously stored contained pointer data, the pointer data was not preserved.

Length of output data buffer
INPUT; BINARY(4)
The length of the output data buffer, in bytes. If this value is larger than the actual size of the output data buffer, the results may not be predictable.

Qualified program name
INPUT; CHAR(20)
The program object for which you want to retrieve data from the associated space. This must be an OPM, user-state program in the user domain. An error is returned if the program is an Integrated Language Environment

(ILE) program, or if it is a system-state or system-domain program. The first 10 characters contain the program name, and the second 10 characters contain the library name. You can use the following special value for the program name:

* Use a program in the call stack. The call stack counter contains the number of programs up the stack from the calling program to look for the program from which to retrieve data.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Call stack counter
INPUT; BINARY(4)
A number greater than zero identifying the location in the call stack for the program if * is specified for the program name. This number is relative to the program that called this API. This parameter is ignored if the program name is not *.

Data handle
INPUT; CHAR(16)
The identifier to retrieve the information from in the associated space. Specify the data handle that was previously used on the Store Program Associated Space (QCLSPGAS) API to store information in the associated space. If no data was previously stored at this data handle, no error is returned. The length of data available parameter will be set to 0 indicating no data was available.

Length of data available
OUTPUT; BINARY(4)
Either the length of the data actually returned or the length of data available. The possible values follow:

0 No data was previously stored at the data handle specified.
From 1 to the value of the length of output data buffer parameter
The length of information available and successfully returned in the output data buffer.
Greater than the value of the length of output data buffer parameter
The total length of data available. If the output data buffer is too small to hold all of the information available, the information is truncated to fit the available 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

CPF0301 E Length of data buffer is not valid.
CPF0302 E Value for call stack counter not valid.
CPF0304 E Operation not allowed on program &1 in library &2.
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.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9820 E Not authorized to use library &1.
CPF9821 E Not authorized to program &1 in library &2.
CPF9830 E Cannot assign library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Program Information (QCLRPGMI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified program name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Program Information (QCLRPGMI) API lets you retrieve program information and place it into a single variable in the calling program. The amount of information returned is limited to the size of the variable. This information is the same as the information returned using the Display Program (DSPPGM) command.

You can use the QCLRPGMI API to retrieve the following:

- Program creation information
- Program statistics
- Program performance information
- SQL/400 statement information
- ILE program size information

Authorities and Locks

Library Authority *READ
Program Authority *READ
Program Lock *SHRRD

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. You can specify the size of this area to be smaller than the

format requested as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If this value is larger than the actual size of the receiver variable, the results may not be predictable. The minimum value is 8.

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

PGMI0100 Basic program information for OPM and ILE programs. For more information, see "PGMI0100 Format."
PGMI0200 Basic program information for OPM and ILE programs plus SQL/400 statement information for OPM programs. For more information, see "PGMI0200 Format" on page 52-39.
PGMI0300 ILE program size information. For more information, see "PGMI0300 Format" on page 52-40.

Qualified program name
INPUT; CHAR(20)
The first 10 characters contain the program name. The second 10 characters contain the name of the library where the program is located. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

PGMI0100 Format

The following information is returned for the PGMI0100 format. Some of the fields returned are blank or zero if they do not apply to the type of program specified. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 52-40.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
Program creation information			
8	8	CHAR(10)	Program name
18	12	CHAR(10)	Program library name
28	1C	CHAR(10)	Program owner
38	26	CHAR(10)	Program attribute
48	30	CHAR(13)	Creation date and time
61	3D	CHAR(10)	Source file name
71	47	CHAR(10)	Source file library name
81	51	CHAR(10)	Source file member name
91	5B	CHAR(13)	Source file updated date and time
104	68	CHAR(1)	Observable information
105	69	CHAR(1)	User profile option
106	6A	CHAR(1)	Use adopted authority
107	6B	CHAR(1)	Log commands
108	6C	CHAR(1)	Allow RTVCLSRC
109	6D	CHAR(1)	Ignore decimal data
110	6E	CHAR(50)	Text description
160	A0	CHAR(1)	Type of program
161	A1	CHAR(59)	Reserved
Program statistics information			
220	DC	BINARY(4)	Minimum number of parameters
224	E0	BINARY(4)	Maximum number of parameters
228	E4	BINARY(4)	Program size
232	E8	BINARY(4)	Associated space size
236	EC	BINARY(4)	Static storage size
240	F0	BINARY(4)	Automatic storage size
244	F4	BINARY(4)	Number of MI instructions
248	F8	BINARY(4)	Number of MI ODT entries
252	FC	CHAR(1)	Program state
253	FD	CHAR(14)	Compiler identification
267	10B	CHAR(6)	Earliest release program can run
273	111	CHAR(10)	Sort sequence table name
283	11B	CHAR(10)	Sort sequence table library name
293	125	CHAR(10)	Language identifier
303	12F	CHAR(1)	Program domain
304	130	CHAR(21)	Reserved
Program performance information			
325	145	CHAR(1)	Optimization
326	146	CHAR(1)	Paging pool
327	147	CHAR(1)	Update program automatic storage area (PASA)
328	148	CHAR(1)	Clear program automatic storage area (PASA)
329	149	CHAR(1)	Paging amount

Offset		Type	Field
Dec	Hex		
330	14A	CHAR(18)	Reserved
ILE information			
348	15C	CHAR(10)	Program entry procedure module
358	166	CHAR(10)	Program entry procedure module library
368	170	CHAR(30)	Activation group attribute
398	18E	CHAR(1)	Observable information compressed
399	18F	CHAR(1)	Run-time information compressed
400	190	CHAR(6)	Release program created on
406	196	CHAR(2)	Reserved
408	198	BINARY(4)	Program CCSID
412	19C	BINARY(4)	Number of modules
416	1A0	BINARY(4)	Number of service programs
420	1A4	BINARY(4)	Number of copyrights

PGMI0200 Format

The following information is returned for the PGMI0200 format. Some of the fields returned are blank or zero if they do not apply to the type of program specified. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 52-40.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
Program creation information			
8	8	CHAR(10)	Program name
18	12	CHAR(10)	Program library name
28	1C	CHAR(10)	Program owner
38	26	CHAR(10)	Program attribute
48	30	CHAR(13)	Creation date and time
61	3D	CHAR(10)	Source file name
71	47	CHAR(10)	Source file library name
81	51	CHAR(10)	Source file member name
91	5B	CHAR(13)	Source file updated date and time
104	68	CHAR(1)	Observable information
105	69	CHAR(1)	User profile option
106	6A	CHAR(1)	Use adopted authority
107	6B	CHAR(1)	Log commands
108	6C	CHAR(1)	Allow RTVCLSRC
109	6D	CHAR(1)	Ignore decimal data
110	6E	CHAR(50)	Text description
160	A0	CHAR(1)	Type of program
161	A1	CHAR(59)	Reserved
Program statistics information			
220	DC	BINARY(4)	Minimum number of parameters
224	E0	BINARY(4)	Maximum number of parameters
228	E4	BINARY(4)	Program size
232	E8	BINARY(4)	Associated space size
236	EC	BINARY(4)	Static storage size
240	F0	BINARY(4)	Automatic storage size
244	F4	BINARY(4)	Number of MI instructions
248	F8	BINARY(4)	Number of MI ODT entries
252	FC	CHAR(1)	Program state
253	FD	CHAR(14)	Compiler identification
267	10B	CHAR(6)	Earliest release program can run
273	111	CHAR(10)	Sort sequence table name
283	11B	CHAR(10)	Sort sequence table library name
293	125	CHAR(10)	Language identifier
303	12F	CHAR(1)	Program domain
304	130	CHAR(21)	Reserved
Program performance information			
325	145	CHAR(1)	Optimization
326	146	CHAR(1)	Paging pool
327	147	CHAR(1)	Update program automatic storage area (PASA)
328	148	CHAR(1)	Clear program automatic storage area (PASA)
329	149	CHAR(1)	Paging amount
330	14A	CHAR(18)	Reserved
Program SQL information			
348	15C	BINARY(4)	Number of SQL statements
352	160	CHAR(18)	Relational database
370	172	CHAR(10)	Commitment control
380	17C	CHAR(10)	Allow copy of data
390	186	CHAR(10)	Close SQL cursor
400	190	CHAR(10)	Naming convention
410	19A	CHAR(10)	Date format
420	1A4	CHAR(1)	Date separator
421	1A5	CHAR(10)	Time format
431	1AF	CHAR(1)	Time separator

Offset		Type	Field
Dec	Hex		
432	1B0	CHAR(10)	Delay PREPARE
442	1BA	CHAR(10)	Allow blocking
ILE information			
452	1C4	CHAR(10)	Program entry procedure module
462	1CE	CHAR(10)	Program entry procedure module library
472	1D8	CHAR(30)	Activation group attribute
502	1F6	CHAR(1)	Observable information compressed
503	1F7	CHAR(1)	Run-time information compressed
504	1F8	CHAR(6)	Release program created on
510	1FE	CHAR(2)	Reserved
512	200	BINARY(4)	Program CCSID
516	204	BINARY(4)	Number of modules
520	208	BINARY(4)	Number of service programs
524	20C	BINARY(4)	Number of copyrights

PGMI0300 Format

The following information is returned for the PGMI0300 format. This format is valid only for ILE programs. If an OPM program is specified, no data is returned and an error is returned. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
ILE program size information			
8	8	CHAR(10)	Program name
18	12	CHAR(10)	Program library name
28	1C	BINARY(4)	Current total program size
32	20	BINARY(4)	Maximum program size
36	24	BINARY(4)	Current number of modules
40	28	BINARY(4)	Maximum number of modules
44	2C	BINARY(4)	Current number of service programs
48	30	BINARY(4)	Maximum number of service programs
52	34	BINARY(4)	Current string directory size
56	38	BINARY(4)	Maximum string directory size
60	3C	BINARY(4)	Current copyright string size
64	40	BINARY(4)	Maximum copyright string size
68	44	BINARY(4)	Current number of auxiliary storage segments
72	48	BINARY(4)	Maximum number of auxiliary storage segments
76	4C	BINARY(4)	Minimum static storage size
80	50	BINARY(4)	Maximum static storage size

Field Descriptions

For more detailed information than that provided in the following field descriptions, refer to documentation for the command that was used to create the program. For information on non-SQL fields, this would normally be one of the following:

- One of the create program (CRTxxxPGM) commands described in the programmer's guide for the language, identified by the xxx in the command name.
- The Create Program (CRTPGM) command.

For information on SQL fields, this would normally be a command of the form CRTSQLxxx described in the *SQL/400* Programmer's Guide*. The xxx in the command name identifies the base language (RPG, COBOL, and so on) of the program.

Activation group attribute. The activation group attribute of this ILE program. The possible values are:

**NEW* A new activation group with the same name as the program name is created when this program is called. The program runs in this activation group.

**DFTACTGRP* The program uses one of two existing activation groups created when the process is started. One default activation group is reserved for system-state programs. The other default activation group is reserved for user-state programs.

**CALLER* The program runs in the activation group of the program from which it is called.

activation group name The name of the activation group in which this program runs. If the activation group already exists when the program is called, the program runs in the existing activation group. If the activation group does not exist when the program is called, a new activation group is created and the program runs in it.

Blank This program is an OPM program.

Allow blocking. Whether blocking is used to improve the performance of certain SQL statements. The possible values are:

**NONE*	Blocking is not used.
**READ*	Blocking is used for read-only data cursors when running COMMIT(*NONE) and there are no EXECUTE or EXECUTE IMMEDIATE statements.
**ALLREAD*	Blocking is used for all read-only cursors when running COMMIT(*NONE) or COMMIT(*CHG).
Blank	The program does not contain SQL statements or it is an ILE program.

Allow copy of data. Whether a copy of the data can be used in the implementation of an SQL query. Possible values are:

**NO*	A copy of the data cannot be used.
**YES*	A copy of the data can be used when needed.
**OPTIMIZE*	The system determines whether a copy of the data is used for optimal performance.
Blank	The program does not contain SQL statements, or it is an ILE program.

Allow RTVCLSRC. The compiler allowed you to control this attribute through the ALWRTVSRC parameter if this program was created using the Create CL Program (CRTCLPGM) command. The possible values are:

N	Source for the CL program is not saved with the program (*NO).
Y	Source is saved (*YES).

Source that is saved can be retrieved by using the Retrieve CL Source (RTVCLSRC) command. This information is blank if the program is not a CL program.

Associated space size. The size (in bytes) of the associated space used by this program.

Automatic storage size. The size (in bytes) of the automatic storage used by this program. This information is blank if the program is an ILE program.

Bytes available. The length of all data available for the requested format. All available data is returned if enough space is provided.

Bytes returned. The length of the data actually returned.

Clear program automatic storage area (PASA). The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program. The possible values are:

N	Do not clear PASA storage (*NOCLRPASA).
C	Clear PASA storage (*CLRPASA).
Blank	The program is an ILE program.

*NOCLRPASA reduces the time needed to call the program. However, if a program variable is referred to before it has been set, it may contain unpredictable data.

*CLRPASA increases the time needed to call the program. However, it ensures that if a program variable is referred to before it has been set, it will contain binary zeros instead of unpredictable data.

Close SQL cursor. Specifies when SQL cursors are implicitly closed and SQL-prepared statements are implicitly discarded. Possible values are:

**ENDPGM*	When the program that contains the SQL statements ends. This value is valid for OPM programs only.
**ENDSQL*	When the last program containing SQL statements ends. This value is valid for OPM programs only.
**ENDJOB*	When the job ends. This value is valid for OPM programs only.
**ENDMOD*	When the module ends. This value is valid for ILE programs only.
**ENDACTGRP*	When the activation group is deleted. This value is valid for ILE programs only.
Blank	The program does not contain SQL statements.

Commitment control. The level of commitment control that was specified on the SQL precompile. Possible values are:

**NONE*	No commitment control was specified on the SQL precompile. Uncommitted changes in other jobs can be seen.
**CHG*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). Updated, deleted, and inserted rows (records) are locked until the end of the unit of work. Uncommitted changes in other jobs can be seen.
**CS*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). Updated, deleted, and inserted rows (records) are locked until the end of the unit of work. A row (record) that is selected but not updated is locked until the next row (record) is selected. Uncommitted changes in other jobs cannot be seen.
**ALL*	Objects referred to in SQL COMMENT ON, CREATE, DROP, GRANT, LABEL ON, and REVOKE statements are locked until the end of the unit of work (transaction). All rows selected, updated, deleted, and inserted are locked until the end of the unit of work. Uncommitted changes in other jobs cannot be seen.
Blank	The program does not contain SQL statements, or it is an ILE program.

Compiler identification. The licensed program identifier, version, release, and modification level of the compiler. The field has a ppppppbVvRrMm format, where:

ppppppp	The licensed program identifier.
b	A blank character.
Vv	The character V is followed by a 1-character version number.
Rr	The character R is followed by a 1-character release level.

| *Mm* The character M is followed by a 1-character modification level.

For programs created by the Create Program (QPRCRTPG) API, this field identifies the version of the operating system that the program was created under.

The field may be blank if the program is created without going through a compilation process or if it is an ILE program.

Creation date and time. The date and time the program was created. The creation date and time field is in the CYYMMDDHHMMSS format, where:

| *C* Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
| *YY* Year
| *MM* Month
| *DD* Day
| *HH* Hour
| *MM* Minute
| *SS* Second

Current copyright string size. The ILE program's copyright string size.

Current number of auxiliary storage segments. The number of auxiliary storage segments in this ILE program.

Current number of modules. The number of modules bound into this ILE program.

Current number of service programs. The number of service programs bound to this ILE program.

Current string directory size. The ILE program's string directory size.

Current total program size. The total size of the ILE program, in kilobytes.

Date format. The format used when accessing date-result columns through SQL. All output date fields are returned in this format. For input date strings, the value you specify is used to determine whether the date is a valid format. Possible values are:

| **USA* USA format (mm/dd/yyyy).
| **ISO* International Standards Organization format (yyyy-mm-dd).
| **EUR* European format (dd.mm.yyyy).
| **JIS* Japanese Industrial Standard Christian Era (yyyy-mm-dd).
| **MDY* Month/day/year format (mm/dd/yy).
| **DMY* Day/month/year format (dd/mm/yy).
| **YMD* Year/month/day format (yy/mm/dd).
| **JUL* Julian format (a numeric value from 1 to 365).
| *blank* The program does not contain SQL statements, or it is an ILE program.

Date separator. The separator used when accessing date-result columns. This information is blank if the program does not contain SQL statements or if it is an ILE program.

However, the number of SQL statements field should be checked to determine if the program contains SQL statements. This is because a blank may be specified as a separator value.

Delay PREPARE. Whether SQL prepare processing can be delayed until the statement is actually used. Possible values are:

| **YES* Prepare processing can be delayed.
| **NO* Prepare processing cannot be delayed.
| *Blank* The program does not contain SQL statements, or it is an ILE program.

Earliest release program can run. The version, release, and modification level of the earliest release the program is allowed to run on. The compiler may have allowed you to control this through the TGTRLS parameter of the command used to create the program. The field has a VvRrMm format, where:

| *Vv* The character V is followed by a 1-character version number.
| *Rr* The character R is followed by a 1-character release level.
| *Mm* The character M is followed by a 1-character modification level.

Ignore decimal data. The compiler may have allowed you to control this attribute through the ignore decimal data error (IGNDECERR) parameter of the command used to create the program. The possible values are:

| *N* An error is signaled to the program without correcting the data that is not valid (*NO).
| *Y* Decimal data that is not valid is corrected (*YES).
| *Blank* The program is an ILE program.

Language identifier. Returns the 3-character language identifier used when the program was compiled. The following special values can also be returned:

| **JOBRUN* The language identifier associated with the job at the time the program is run.
| *Blank* The program does not contain any language identification information.

Log commands. The value specified for the LOG parameter of the CRTCLPGM command. This field is meaningful only if the program is a CL program. The possible values are N (*NO), Y (*YES), and J (*JOB). This information is blank if the program is not a CL program.

Maximum copyright string size. The maximum size of the the copyright string in an ILE program.

Maximum number of auxiliary storage segments. The maximum number of auxiliary storage segments an ILE program can have.

Maximum number of modules. The maximum number of modules that can be bound into an ILE program.

Maximum number of parameters. The maximum number of parameters that may be received by the program when it is called. A value of -1 is returned if the program is not observable.

Maximum number of service programs. The maximum number of service programs that can be bound to an ILE program.

Maximum program size. The maximum size that an ILE program can be, in kilobytes.

Maximum static storage size. The maximum static storage size (in bytes) that this program may need in order to run.

Maximum string directory size. The maximum size that the string directory can be in an ILE program.

Minimum number of parameters. The minimum number of parameters that is to be received by the program when it is called. A value of -1 is returned if the program is not observable.

Minimum static storage size. The minimum static storage size in bytes that this program needs in order to run.

Naming convention. The convention used for naming objects in SQL statements. Possible values are:

**SQL*	The SQL naming convention is used.
**SYS*	The system naming convention is used.
Blank	The module does not contain SQL statements.

Number of copyrights. The number of copyrights in this ILE program. This field is zero if the program is an OPM program. Do not assume that a value of zero indicates that the program is an OPM program. ILE programs may not have any copyrights, so this value could be zero for an ILE program. Check the type of program field to determine whether the program is an OPM program or an ILE program.

Number of MI instructions. The number of machine interface (MI) instructions used by this program. A value of -1 is returned if the program is not observable. This information is zero if the program is an ILE program.

Number of MI ODT entries. The number of ODT (object definition table) entries for the program. This is the number of program objects declared by the compiler. Program objects include variables, constants, labels, operand lists, and exception descriptions. Typically, one or more ODT entries are used for each variable, constant, and label in your program. Some are used by the compiler for internal purposes. The number of internal ODT entries varies depending on the size and complexity of the program. There is a limit of 32767 ODT entries in a program. A value of -1 is returned if the program is not observable. This information is zero if the program is an ILE program.

Number of modules. The number of modules bound into this program. This information is zero if the program is an OPM program.

Number of service programs. The number of service programs bound to this program. This information is zero if the program is an OPM program. Do not assume that a value of zero indicates that the program is an OPM program. ILE programs may not have any service programs bound to them, so this value could be zero for an ILE program. Check the type of program field to determine whether the program is an OPM program or an ILE program.

Number of SQL statements. The number of SQL/400 statements contained in the program. This value is zero if the program does not contain SQL statements or if it is an ILE program.

Observable information. Whether the observable information associated with the program exists. The possible values are:

A	The observable information exists (*ALL).
N	The observable information does not exist (*NONE).
Blank	The program is an OPM program.

The observable information for most programs may be removed by using the remove observability (RMVOBS) parameter on the Change Program (CHGPGM) command.

Observable information compressed. Whether the observable information associated with the program is compressed. The possible values are:

Y	The observable information is compressed.
N	The observable information is not compressed.
Blank	The program is an OPM program.

Optimization. Indicates what was specified on the OPTIMIZE parameter when the program was created or changed. Possible values are:

N	*NOOPTIMIZE was specified.
O	*OPTIMIZE was specified.
Blank	The program is an ILE program.

Paging amount. The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program. The possible values are:

N	Page the program one page at a time (*NOBLOCK).
B	Page the program in eight-page blocks (*BLOCK).

*BLOCK gives better performance in most situations. It is likely that more than one page in the block is referred to before being replaced by some other paging occurring in the storage pool.

*NOBLOCK can give better performance if the other pages that would have been brought in as a block are unlikely to be referred to before being replaced by some other paging.

Paging pool. The paging pool used for the program object. The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program. The values returned are:

U	Use the user pool (*USER).

| *B* Use the base pool (*BASE).
| *M* Use the machine pool (*MACHINE).

| *USER is used by most system programs and all user programs, unless GENOPT(*MACHINE) is specified.

| *BASE is used by certain system programs to avoid disturbing the user pool when they need to be paged in. These programs are not used frequently enough to belong in the machine pool. This value will only appear for OPM programs.

| *MACHINE is used by a small number of system programs that are so highly used that their pages should remain almost constantly in main storage. The machine pool is intended to be a stable, low paging pool. If user programs page in the machine pool, there may be contention for main storage between system and user programs. This may adversely affect system performance and response times. To prevent paging contention, increase the QMCHPOOL system value with the Change System Value (CHGSYSVAL) command.

| **Program attribute**. The language the program is written in. (For example, the value is CLP for a CL program, and the value is RPG for an RPG program). This field can be blank. (For example, the program was created by the Create Program (QPRCRTPG) API or the program is created by a compilation process internal to IBM).

| **Program CCSID**. The coded character set identifier (CCSID) for this ILE program. This is 65535 for ILE programs. This information is zero if the program is an OPM program.

| **Program domain**. The domain of the program. The possible values are:

| *S* The program can be called by system-state programs.
| *U* The program can be called by user- or system-state programs.

| **Program entry procedure module**. The module name that contains the program entry procedure for this program. This information is blank if the program is an OPM program.

| **Program entry procedure module library**. The library name that contained the module that contained the program entry procedure for this program when the bind was done. This information is blank if the program is an OPM program.

| **Program library name**. The name of the library containing the program.

| **Program name**. The name of the program.

| **Program owner**. The name of the program owner's user profile.

| **Program size**. The size (in bytes) of this program.

| **Program state**. The state of the program. The possible values are:

| *I* The program runs under (inherits) the same state as its caller.
| *S* The program runs as a system-state program.
| *U* The program runs as a user-state program.

| **Relational database**. The default relational database that was specified on the SQL precompile. A nonblank value other than *LOCAL specifies the name of the relational database to be resolved through the relational database directory. The following special values can be returned:

| **LOCAL* The module can only access data on the local system.
| *Blank* The program does not contain SQL statements, or it is an ILE program.

| **Release program created on**. The version, release, and modification level of the operating system on which the program was created.

| **Reserved**. An ignored field.

| **Run-time information compressed**. Whether the run-time information associated with the program is compressed. The possible values are:

| *Y* The run-time information is compressed.
| *N* The run-time information is not compressed.
| *Blank* The program is an OPM program.

| **Sort sequence table name**. The name of the sort sequence table used when the program was compiled. This does not apply to SQL statements in the module. The following special values can be returned:

| **HEX* No sort sequence is used.
| **JOBRUN* The sort sequence value associated with the job at the time the program is run is used.
| **LANGIDSHR* The shared sort sequence for the language identifier is used.
| **LANGIDUNQ* The unique sort sequence for the language identifier is used.

| **Sort sequence table library name**. The name of the library that contained the sort sequence table used when the program was compiled. This information is blank if:

- The program does not contain any sort sequence information, as is the case for an ILE program.
- A special value was returned for the sort sequence table name.

| The following special values can also be returned:

| **LIBL* The sort sequence table is found by looking for it in the library list when the program is run.
| **CURLIB* The sort sequence table is found in the current library when the program is run.

| **Source file library name**. The name of the library that contains the source file used to create the program. The field is blank if a source file was not used to create the program or if it is an ILE program.

Source file member name. The name of the member in the source file. The field is blank if a source file was not used to create the program or if it is an ILE program.

Source file name. The name of the source file used to create the program. The field is blank if a source file was not used to create the program or if it is an ILE program.

Source file updated date and time. The date and time the member in the source file was last updated. The field is in the same format as the creation time and date. The field is blank if a source file was not used to create the program or if it is an ILE program.

Static storage size. For OPM programs this is the size (in bytes) of the static storage used by the program. For ILE programs this is the maximum amount of static storage (in bytes) that may be needed to run the program.

Text description. The user text, if any, used to briefly describe the program and its function.

Time format. The format used when accessing time-result columns through SQL. All output time fields are returned in this format. Possible values are:

*USA	USA format (hh:mm a.m. or p.m.).
*ISO	International Standards Organization format (hh.mm.ss).
*EUR	European format (hh.mm.ss).
*JIS	Japanese Industrial Standard Christian Era (hh.mm.ss).
*HMS	Hours/minutes/seconds format (hh:mm:ss).
Blank	The program does not contain SQL statements.

Time separator. The separator used when accessing time-result columns. This information is blank if the program does not contain SQL statements. However, the number of SQL statements field should be checked to determine if the program contains SQL statements. This is because a blank may be specified as a separator value.

Type of program. Whether the program is an ILE program or an OPM program. The possible values are:

Blank	OPM program
B	ILE program

Update program automatic storage area (PASA). The compiler may have allowed you to control this attribute through the GENOPT parameter of the command used to create the program. This information is blank if the program is an ILE program. The possible values are:

N	Do not update internal PASA stack information (*NOUPDPASA).
U	Update internal PASA stack information (*UPDPASA).

*NOUPDPASA reduces the time needed to call the program.

*UPDPASA increases the time needed to call the program but is used by some system programs that are dependent on updates of internal PASA stack information.

Use adopted authority. The value specified for the USEADPAUT option on the command used to change the program. The possible values are:

Y	Uses program adopted authority from previous call levels when this program is running (*YES).
N	Does not use program adopted authority from previous call levels when this program is running (*NO).

User profile option. The value specified for the USRPRF option on the command used to create the program. The possible values are:

U	The program runs under the current user's user profile (*USER).
O	The program runs under both the current user's and the owner's user profiles (*OWNER).

Error Messages

CPF5CF5 &1 in library &2 not ILE program.
CPF2150 E Object information function failed.
CPF2151 E Operation failed for &2 in &1 type *&3.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
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.
CPF8122 E &8 damage on library &4.
CPF8123 E Damage on object information for library &4.
CPF8129 E Program &4 in &9 damaged.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9806 E Cannot perform function for object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9820 E Not authorized to use library &1.
CPF9821 E Not authorized to program &1 in library &2.
CPF9830 E Cannot assign library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Service Program Information (QBNRSPGM) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified service program name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Service Program Information (QBNRSPGM) API lets you retrieve service program information and place it into a single variable in the calling program. The amount of information returned is limited to the size of the variable. This information is similar to the information returned using the Display Service Program (DSPSRVPGM) command.

You can use the QBNRSPGM API to retrieve the following:

- Service program creation information
- Service program statistics
- Service program performance information
- Service program size information

Authorities and Locks

Library Authority *READ
Service Program Authority (see note) *READ
Service Program Lock *SHRRD

Note: You must have *USE service program authority to retrieve the following fields in the SPGI0100 format:

- Export source file name
- Export source file library name
- Export source file member name

If you attempt to retrieve these fields with *READ service program authority, they are set to blanks. All other fields in the SPGI0100 format require *READ service program authority.

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. You can specify the size of this area to be smaller than the format requested if you specify the length of receiver variable parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If this value is larger than the actual size of the receiver variable, the results may not be predictable. The minimum value is 8.

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

SPGI0100 Basic service program information. For more information, see "SPGI0100 Format."
SPGI0200 Service program size information. For more information, see "SPGI0200 Format" on page 52-47.

Qualified service program name
INPUT; CHAR(20)
The first 10 characters contain the service program name. The second 10 characters contain the name of the library where the service program is located. You can use these special values for the library name:

**CURLIB* The job's current library
**LIBL* The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

SPGI0100 Format

The following information is returned for the SPGI0100 format. See "Authorities and Locks" for the authority needed regarding specific fields. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 52-47.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Service program name
18	12	CHAR(10)	Service program library name
28	1C	CHAR(10)	Service program owner
38	26	CHAR(10)	Service program attribute
48	30	CHAR(13)	Creation date and time
61	3D	CHAR(10)	Export source file name
71	47	CHAR(10)	Export source file library name
81	51	CHAR(10)	Export source file member name
91	5B	CHAR(30)	Activation group attribute
121	79	CHAR(16)	Current export signature
137	89	CHAR(1)	User profile
138	8A	CHAR(1)	Observable information compressed
139	8B	CHAR(1)	Run-time information compressed

Offset			
Dec	Hex	Type	Field
140	8C	BINARY(4)	Service program CCSID
144	90	BINARY(4)	Number of modules
148	94	BINARY(4)	Number of service programs
152	98	BINARY(4)	Number of copyrights
156	9C	CHAR(50)	Text description
206	CE	CHAR(100)	Reserved
306	132	CHAR(1)	Service program state
307	133	CHAR(1)	Service program domain
308	134	BINARY(4)	Associated space size
312	138	BINARY(4)	Static storage size
316	13C	BINARY(4)	Service program size
320	140	CHAR(6)	Release service program created on
326	146	CHAR(6)	Earliest release service program can run
332	14C	CHAR(100)	Reserved
432	1B0	CHAR(1)	Paging pool
433	1B1	CHAR(1)	Paging amount

SPGI0200 Format

The following information is returned for the SPGI0200 format. For detailed descriptions of the fields in the table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Service program name
18	12	CHAR(10)	Service program library name
28	1C	BINARY(4)	Current total service program size
32	20	BINARY(4)	Maximum service program size
36	24	BINARY(4)	Current number of modules
40	28	BINARY(4)	Maximum number of modules
44	2C	BINARY(4)	Current number of service programs
48	30	BINARY(4)	Maximum number of service programs
52	34	BINARY(4)	Current string directory size
56	38	BINARY(4)	Maximum string directory size
60	3C	BINARY(4)	Current copyright string size
64	40	BINARY(4)	Maximum copyright string size
68	44	BINARY(4)	Current number of auxiliary storage segments
72	48	BINARY(4)	Maximum number of auxiliary storage segments
76	4C	BINARY(4)	Current number of program procedure exports
80	50	BINARY(4)	Maximum number of program procedure exports
84	54	BINARY(4)	Current number of program data exports
88	58	BINARY(4)	Maximum number of program data exports
92	5C	BINARY(4)	Current number of signatures
96	60	BINARY(4)	Maximum number of signatures
100	64	BINARY(4)	Minimum static storage size
104	68	BINARY(4)	Maximum static storage size

Field Descriptions

For more detailed information than that provided in the following field descriptions, refer to the *CL Reference* for the Create Service Program (CRTSRVPGM) command.

Activation group attribute. The activation group attribute of this service program. The possible values are:

**CALLER* The service program runs in the activation group of the program from which it is called.

activation group name
The name of the activation group in which this service program runs. If the activation group already exists when the service program is called, the service program runs in the existing activation group. If the activation group does not exist when the service program is called, a new activation group is created in which the service program runs.

Associated space size. The size (in bytes) of the associated space used by this service program.

Bytes available. The length of all data available for the requested format. All available data is returned if enough space is provided.

Bytes returned. The length of the data actually returned.

Creation date and time. The date and time the service program was created. The creation date and time field is in the CYYMMDDHHMMSS format, where:

| *C* Century. 0 indicates the twentieth century, and 1 indicates the twenty-first century.
| *YY* Year
| *MM* Month
| *DD* Day
| *HH* Hour
| *MM* Minute
| *SS* Second

| **Current copyright string size**. The size of the service program's copyright string.

| **Current export signature**. The current export signature of this service program.

| **Current number of auxiliary storage segments**. The number of auxiliary storage segments in this service program.

| **Current number of modules**. The number of modules bound into this service program.

| **Current number of program data exports**. The number of data items exported from this service program.

| **Current number of program procedure exports**. The number of procedures exported from this service program.

| **Current number of service programs**. The number of service programs bound to this service program.

| **Current number of signatures**. The number of signatures for this service program.

| **Current string directory size**. The service program's string directory size.

| **Current total service program size**. The total size of the service program, in kilobytes.

| **Earliest release service program can run**. The version, release, and modification level of the earliest release on which the service program is allowed to run. The field has a VvRrMm format, where:

| *Vv* The character V is followed by a 1-character version number.
| *Rr* The character R is followed by a 1-character release level.
| *Mm* The character M is followed by a 1-character modification level.

| **Export source file library name**. The name of the library that contains the export source file. This may be blank if:

- No export source file was specified on the CRTSRVPGM command.
- You attempt to retrieve this field with a service program authority of *READ. A service program authority of *USE is required.

| **Export source file member name**. The name of the member in the export source file that was used to create this service program. This may be blank if:

- No export source file was specified on the CRTSRVPGM command.
- You attempt to retrieve this field with a service program authority of *READ. A service program authority of *USE is required.

| **Export source file name**. The name of the export source file that contains the export source file member. This may be blank if:

- No export source file was specified on the CRTSRVPGM command.
- You attempt to retrieve this field with a service program authority of *READ. A service program authority of *USE is required.

| **Maximum copyright string size**. The maximum size of the copyright string in a service program.

| **Maximum number of auxiliary storage segments**. The maximum number of auxiliary storage segments a service program can have.

| **Maximum number of modules**. The maximum number of modules that can be bound into a service program.

| **Maximum number of program data exports**. The maximum number of data items that can be exported by a service program.

| **Maximum number of program procedure exports**. The maximum number of procedures that can be exported by a service program.

| **Maximum number of service programs**. The maximum number of service programs that can be bound to a service program.

| **Maximum number of signatures**. The maximum number of signatures that can be in a service program.

| **Maximum service program size**. The maximum size of a service program, in kilobytes.

| **Maximum static storage size**. The maximum amount of static storage in bytes that may be needed to run the service program.

| **Maximum string directory size**. The maximum size of the string directory in a service program, in bytes.

| **Minimum static storage size**. The minimum amount of static storage, in bytes, needed to run the service program.

| **Number of copyrights**. The number of copyrights in this service program.

| **Number of modules**. The number of modules bound into this service program.

| **Number of service programs**. The number of service programs bound to this program.

| **Observable information compressed**. Whether the observable information associated with the service program is compressed. The possible values are:

| *Y* The observable information is compressed.
| *N* The observable information is not compressed.

| **Paging amount**. The compiler may have allowed you to control this attribute through the GENOPT parameter of the CRTSRVPGM command. The possible values are:

| *N* Page the program one page at a time (*NOBLOCK).
| *B* Page the program in eight-page blocks (*BLOCK).

| *BLOCK gives better performance in most situations. It is likely that more than one page in the block is referred to before being replaced by other paging occurring in the storage pool.

| *NOBLOCK can give better performance if the other pages that would have been brought in as a block are unlikely to be referred to before being replaced by other paging.

| **Paging pool**. The paging pool used for the service program object. The compiler may have allowed you to control this attribute through the GENOPT parameter of the CRTSRVPGM command. The values returned are:

| *U* Use the user pool (*USER).
| *M* Use the machine pool (*MACHINE).

| *USER is used by most system service programs and all user service programs unless GENOPT(*MACHINE) is specified.

| *MACHINE is used by a few system service programs that are so highly used that their pages should remain almost constantly in main storage. The machine pool is intended to be a stable, low paging pool. If user service programs page in the machine pool, there may be contention for main storage between system and user service programs. This may adversely affect system performance and response times. To prevent paging contention, increase the QMCHPOOL system value with the Change System Value (CHGSYSVAL) command.

| **Release service program created on**. The version, release, and modification level of the operating system on which the service program was created. The field has a VvRrMm format, where:

| *Vv* The character V is followed by a 1-character version number.
| *Rr* The character R is followed by a 1-character release level.
| *Mm* The character M is followed by a 1-character modification level.

| **Reserved**. An ignored field.

| **Run-time information compressed**. Whether the run-time information associated with the service program is compressed. The possible values are:

| *Y* The run-time information is compressed.
| *N* The run-time information is not compressed.

| **Service program attribute**. The language the program is written in (for example, RPG.)

| **Service program CCSID**. The coded character set identifier (CCSID) for this service program. This is 65535 for service programs.

| **Service program domain**. The domain of the service program. The possible values are:

| *S* The service program can be called by system-state programs.
| *U* The service program can be called by user- or system-state programs.

| **Service program library name**. The name of the library containing the service program.

| **Service program name**. The name of the service program.

| **Service program owner**. The name of the service program owner's user profile.

| **Service program size**. The size (in bytes) of this service program.

| **Service program state**. The state of the service program. The possible values are:

| *I* The service program runs under (inherits) the same state as its caller.
| *S* The service program can call user- or system-state programs.
| *U* The service program can call user-state programs.

| **Static storage size**. The maximum amount of static storage in bytes that may be needed to run the service program.

| **Text description**. The user text, if any, used to briefly describe the service program and its function.

| **User profile**. The value specified for the USRPRF option on the CRTSRVPGM command. The possible values are:

| *U* The service program runs under the current user's user profile (*USER).
| *O* The service program runs under both the current user's and the owner's user profiles (*OWNER).

| Error Messages

| CPF2150 E Object information function failed.
| CPF2151 E Operation failed for &2 in &1 type *&3.
| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3C19 E Error occurred with receiver variable specified.
| CPF3C20 E Error found by program &1.
| CPF3C21 E Format name &1 is not valid.
| CPF3C24 E Length of the receiver variable is not valid.

| CPF8122 E &8 damage on library &4.
| CPF8123 E Damage on object information for library &4.
| CPF813D E Service program &4 in &9 damaged.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9806 E Cannot perform function for object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9811 E Program &1 in library &2 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9821 E Not authorized to program &1 in library &2.
| CPF9830 E Cannot assign library &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Store Program Associated Space (QCLSPGAS) API

Parameters

Required Parameter Group:

1	Input data buffer	Input	Char(*)
2	Length of input data buffer	Input	Binary(4)
3	Qualified program name	Input	Char(20)
4	Call stack counter	Input	Binary(4)
5	Data handle	Input	Char(16)
6	Error code	I/O	Char(*)

The Store Program Associated Space (QCLSPGAS) API allows you to store information in the associated space of an original program model (OPM), user-state program in the user domain. This API is intended to be used only on programs created by the Create Program (QPRCRTPG) API.

This API's primary use is to store information about a program object for later use. For example, it can be used by a compiler to store information about the compilation process that is needed later at run time. The space associated with a program object is not a replacement for data areas or user spaces. It is recommended for static data only.

There is no capability to delete the information after it is stored. Multiple store requests using the same data handle will write over any information previously stored.

Authorities and Locks

Program Library Authority *USE
Program Authority *ALL
Program Lock *EXCL

Required Parameter Group

Input data buffer
INPUT; CHAR(*)
The information to store in the associated space. This information can contain scalar data only. If it does contain pointer data, the pointer data is not kept.

Length of input data buffer
INPUT; BINARY(4)
The length of the data in the input data buffer, in bytes. The maximum size of an associated space is 16MB. An error is returned if the length of the input data and the data already stored in the associated space exceeds 16MB.

Qualified program name
INPUT; CHAR(20)
The program object for which you want to store data in the associated space. This must be an OPM, user-state program in the user domain. An error is returned if the program is an ILE program, or if it is a system-state or system-domain program. The first 10 characters contain the program name, and the second 10 characters contain the library name. You can use the following special value for the program name:

*	A program in the call stack. The call stack counter contains the number of programs up the stack from the calling program to look for the program where the data is to be stored.

You can use the following special values for the library name:

***CURLIB**	The job's current library
***LIBL**	The library list

Call stack counter
INPUT; BINARY(4)
A number greater than zero identifying the location in the call stack for the program if * is specified for the program name. This number is relative to the program that called this API. This parameter is ignored if the program name is not *.

Data handle
INPUT; CHAR(16)
The identifier to store the information under in the associated space. This identifier can be any unique value you choose to store your information under. You must remember this value if you want to later retrieve this information. Data may have been stored under a data handle. If this API is called to store data under it, the existing data is overwritten with the new 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.

Error Messages

| CPF0301 E Length of data buffer is not valid.
| CPF0302 E Value for call stack counter not valid.
| CPF0303 E Limit of associated space size exceeded.
| CPF0304 E Operation not allowed for program &1 in library &2.
| 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.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list previously deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9811 E Program &1 in library &2 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9821 E Not authorized to program &1 in library &2.
| CPF9830 E Cannot assign library &1.
| CPF9838 E User profile storage limit exceeded.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 19. Security APIs

Chapter 53. Security APIs

The OS/400 security APIs let you combine many individual jobs into a single server or overhead job without compromising system security. These APIs can be used to consolidate server jobs to reduce processing time and storage use because the system performs job management tasks for only one job. It also speeds response time for system users.

The APIs in this chapter are presented in alphabetical order.

The security APIs and their functions follow:

- **Change Previous Sign-On Date** (QSCHGPR) changes the previous sign-on date and time to the current date and time for the current user of the job.
- **Change User Password** (QSYCHGPW) changes a user's password.
- **Check User Authority to an Object** (QSYCUSRA) returns an indication about a user's specified authority to an object.
- **Check User Special Authorities** (QSYCUSRS) returns an indication of a user's special authorities.
- **Convert Authority Values to MI Value** (QSYCVTA) converts authority values to the machine interface (MI) representation of the value.
- **Get Profile Handle** (QSYGETPH) validates a user ID and password, and creates an encrypted abbreviation called a profile handle for that user profile.
- **List Authorized Users** (QSYLAUTU) puts a list of authorized users of the system in a user space.
- **List Objects Secured by Authorization List** (QSYLATLO) puts a list of objects secured by an authorization list in a user space.
- **List Objects That Adopt Owner Authority** (QSYLOBJP) puts a list of objects that adopt an owner's authority in a user space.
- **List Objects User Authorized to or Owns** (QSYLOBJA) puts a list of objects that a user is authorized to and/or owns in a user space.
- **List Users Authorized to Object** (QSYLUSRA) puts a list of users privately authorized to an object in a user space.
- **Release Profile Handle** (QSYRLSPH) deletes a profile handle.
- **Retrieve User Authority to Object** (QSYRUSRA) returns the user's authority to an object.
- **Retrieve User Information** (QSYRUSRI) returns the information about a user.
- **Set Profile** (QWTSETP) switches the job to run under a new profile.

For general information about OS/400 system security, see the *Security Reference* manual.

Change Previous Sign-On Date (QSYCHGPR) API

Parameters

Required Parameter:

1	Error Code	I/O	CHAR(*)

The Change Previous Sign-On Date (QSYCHGPR) API changes the previous sign-on date and time to the current date and time for the current user of the job. If a user has been swapped in using the Set Profile (QWTSETP) API, that user's previous sign-on date and time is the one that gets changed.

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

CPF2213 E Not able to allocate user profile &1.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Change User Password (QSYCHGPW) API

Parameters

Required Parameter Group:

1	User	Input	Char(10)
2	Current password	Input	Char(10)
3	New password	Input	Char(10)
4	Error code	I/O	Char(*)

The Change User Password (QSYCHGPW) API changes a user's password. You must know the existing password that you want to change.

This API provides support similar to the Change Password (CHGPWD) command.

Authorities and Locks

If the user parameter is not *CURRENT or the current user for the job, the caller of the API must have *SECADM special authority and *OBJMGT and *USE authorities to the user profile being changed to change the password.

If the caller of the API:

- Enters the wrong password for the user and,
- Exceeds the maximum number of times allowed by the system value QMAXSIGN, and
- The system value QMAXSGNACN is set to disable user profiles,

then the user profile specified on the user parameter is disabled.

You cannot specify the following user profile names for the user parameter:

QDBSHR	QDSNX	QLPAUTO	QSPL
QDFTOWN	QFNC	QLPINSTALL	QSPLJOB
QDOC	QGATE	QSNADS	QSYS
			QTSTRQS

When the new password is checked to ensure it meets the password composition rules for the system, only one error is returned per API call. Therefore, if the new password fails more than one of the rules, multiple calls to the API are needed to determine a correct new password.

You should avoid calling this API from a command line. If this API is called from CL and CL commands are being logged for the job or CL program, the call parameters for the API are logged in the job log. This means the passwords appear in the job log.

Required Parameter Group

User
INPUT; CHAR(10)
The name of the user whose password is being changed. You can specify the following special value:

***CURRENT** The password of the user currently running is changed.

Current password
INPUT; CHAR(10)
The current password for the user. Verification is done to ensure this is the correct password for the user before the password is changed. You can specify the following special value:

***NONE** The user currently does not have a password.

New password
INPUT; CHAR(10)
The new password for the user. Verification is done to ensure the new password meets the password composition rules of the system. You can specify the following special value:

***NONE** The user is changed to not have a password. This value is not allowed if *CURRENT, the current user name for the job, or QSECOFR is specified on the user parameter.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

| CPD2201 E System user profile cannot be changed.
| CPF0001 E Error found on &1 command.
| CPF22C0 E Password does not meet password rules. Return code &1.
| CPF22C2 E Password less than &1 characters.
| CPF22C3 E Password longer than &1 characters.
| CPF22C4 E Password matches one of 32 previous passwords.
| CPF22C5 E Password contains one of the following: &1.
| CPF22C6 E Password contains two numbers next to each other.
| CPF22C7 E Password contains a character used more than once.
| CPF22C8 E Same character in same position as previous password.
| CPF22C9 E Password must contain a number.
| CPF22D1 E Password cannot be same as user ID.
| CPF22D2 E Password approval program &1 not found.
| CPF22D3 E Password approval program signaled an error.
| CPF22D4 E Not allowed to use password approval program.
| CPF22D5 E Parameters in password approval program not correct.
| CPF22E2 E Password not correct for user profile &1.
| CPF22E3 E User profile &1 is disabled.
| CPF22F5 E Value &1 for new password not allowed.
| CPF22F6 E New password cannot be *NONE.
| CPF2203 E User profile &1 not correct.
| CPF2213 E Not able to allocate user profile &1.
| CPF2225 E Not able to allocate internal system object.
| CPF2292 E *SECADM required to create or change user profiles.
| CPF3CF1 E Error code parameter not valid.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Check User Authority to an Object (QSYCUSRA) API

Parameters

Required Parameter Group:

1	Authority indicator	Output	Char(1)
2	Qualified user name	Input	Char(10)
3	Object name	Input	Char(20)
4	Object type	Input	Char(10)
5	Authority	Input	Char(*)
6	Number of authorities	Input	Binary(4)
7	Call level	Input	Binary(4)
8	Error code	I/O	Char(*)

The Check User Authority to Object (QSYCUSRA) API provides an indication of whether the user has the specified authority to an object.

Authorities and Locks

The following authority is required for the user calling this API, unless the user parameter is *CURRENT or the current user for the job, or the caller owns the object, or the object is an authorization list.

- *OBJMGT authority to the object.
- *READ authority to the user profile.

If the user is *CURRENT, or the current user for the job, the authority to the user includes any authority specified on the object (either private, group, authorization list, or public) plus any program adopted authority. If the user specified is not *CURRENT, the authority available to the user is the authority specified on the object; no program adopted authority is used.

Adopted authority is authority given to the user by the program for the duration of that program. If previous programs in the program stack adopt their owner's authority, the adopted authority for the current program is the accumulated adopted authority from all other programs in the program stack that adopt authority.

When the API checks for authority to logical files, and any data authorities (read, add, update, and delete), are specified, the authority check fails, unless the user adopts authority or has *ALLOBJ special authority. Logical files do not have data authorities associated with them, the data authorities are associated with the base physical file.

Required Parameter Group

Authority indicator
OUTPUT; CHAR(1)
Whether the user has the specified authority to the object. The field contains one of the following:

Y The user has the specified authority.
N The user does not have the specified authority.

Qualified user name
INPUT; CHAR(10)
The name of the user whose authority is checked.

You can specify the following special value:

***CURRENT** Checks the authority of the current user to the specified object.

Object name
INPUT; CHAR(20)
The name of the object whose authority is checked. The first 10 characters specify the object name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the object. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the object.

Object type
INPUT; CHAR(10)
The type of object whose authority is checked.

Authority
INPUT; CHAR(*)
The authority to check for. This parameter can contain up to eight 10-character fields. The following identifies the type of authority the user has to the object:

***EXCLUDE** Exclude authority. If this value is specified, no other values can be specified.
***ALL** All authority.
***CHANGE** Change authority.
***USE** Use authority.
***AUTLMGT** Authorization list management authority. This value is only valid if the object type is *AUTL.
***OBJOPR** Object operational authority.
***OBJMGT** Object management authority.
***OBJEXIST** Object existence authority.
***READ** Read authority.
***ADD** Add authority.
***UPD** Update authority.
***DLT** Delete authority.

Number of authorities
INPUT; BINARY(4)
The number of authorities specified in the authority parameter. You can specify 1 through 8 authorities.

Call level
INPUT; BINARY(4)
The number of call levels to back up in the program stack to do the authority check. For example, if the program that calls this API adopts authority, you would probably not want the authority check to use the adopted authority. Therefore, the authority check should be done at the call level previous to the current level. This parameter should then contain a 1. You can check the authority at the various call levels by signifying a numeric equivalent to the call level. For example, to

check the authority at the current call level, specify a 0; to check the authority at the previous call level, specify a 1.

This parameter is only used if the user parameter is *CURRENT or the current user for the job.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

| CPF22FA E Authority value &1 not valid.
| CPF22FB E Must specify *EXCLUDE or *AUTL as only authority value.
| CPF22F7 E Number of authorities must be between 1 and &1.
| CPF22F9 E Call level &1 not valid.
| CPF3CF1 E Error code parameter not valid.
| CPF3C31 E Object type &1 is not valid.
| CPF8122 E &8 damage on library &4.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9811 E Program &1 in library &2 not found.
| CPF9812 E File &1 in library &2 not found.
| CPF9814 E Device &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Check User Special Authorities (QSYCUSRS) API

Parameters

Required Parameter Group:

1	Authority indicator	Output	Char(1)
2	User name	Input	Char(10)
3	Special authority	Input	Char(*)
4	Number of authorities	Input	Binary(4)
5	Call level	Input	Binary(4)
6	Error code	I/O	Char(*)

The Check User Special Authorities (QSYCUSRS) API provides an indication of whether the user has the specified special authorities.

Authorities and Locks

User Profile Authority *READ

When the API checks for special authorities and the user parameter is *CURRENT or the current user for the job, the special authorities available to the user include any special authorities the user or the group has, and any program adopted special authorities. If the user specified is not the user currently running, then the special authorities available to the user are only the special authorities the user and his group have.

If previous programs in the program stack adopt their owner's authority, the adopted authority for the current program is the accumulated adopted authority from all other programs in the program stack that adopt authority.

Required Parameter Group

Authority indicator
OUTPUT; CHAR(1)
Whether the user has the specified special authorities. This parameter contains one of the following:

Y The user has the specified special authorities.
N The user does not have the specified special authorities.

User name
INPUT; CHAR(10)
The name of the user whose special authorities are checked.

You can specify the following special value:

***CURRENT** The special authorities for the user currently running are checked.

Special authority
INPUT; CHAR(*)
| The special authorities checked for the user. This
| parameter can contain up to seven 10-character fields. Each of the 10-character fields can contain one of the following special values.

***ALLOBJ** All object special authority.
| ***AUDIT** Audit special authority.
***SECADM** Security administrator special authority.
***JOBCTL** Job control special authority.
***SPLCTL** Spool control special authority.
***SAVSYS** Save system special authority.
***SERVICE** Service special authority.

Number of authorities
INPUT; BINARY(4)
The number of special authorities specified in the special
| authority parameter. You can specify 1 through 7.

Call level
INPUT; BINARY(4)
The number of call levels to back up in the program stack to do the authority check. For example, if the program that calls this API adopts authority, you would probably not want the authority check to use the adopted

authority. Therefore, the authority check should be done at the call level previous to the current level. This parameter should then contain a 1. You can check the authority at the various call levels by signifying a numeric equivalent to the call level. For example, to check the authority at the current call level, specify a 0; to check the authority at the previous call level, specify a 1.

This parameter is only used if the user parameter is *CURRENT, or the current user name for the job.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

| CPF22F7 E Number of authorities must be between 1 and &1.
| CPF22F8 E Special authority value &1 not valid.
| CPF22F9 E Call level &1 not valid.
| CPF2203 E User profile &1 not correct.
| CPF2225 E Not able to allocate internal system object.
| CPF3CF1 E Error code parameter not valid.
| CPF8122 E &8 damage on library &4.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Convert Authority Values to MI Value (QSYCVTA) API

Parameters

Required Parameter Group:

1	Converted authority value	Output	Char(2)
2	Authority special value	Input	Char(*)
3	Number of authorities	Input	Binary(4)
4	Error code	I/O	Char(*)

The Convert Authority Values to MI Value (QSYCVTA) API converts the special values indicating authority to the corresponding machine interface (MI) representation of that value.

Required Parameter Group

Converted authority value
OUTPUT; CHAR(2)
The MI representation of the authority special value in hexadecimal format.

Authority special value
INPUT; CHAR(*)
The authority special values that are converted. The converted value is the cumulative value of all authority special values specified. This parameter can contain up to eight 10-character fields. Each of the 10-character fields can contain one of the following special values. The following identifies the authority special values that are converted to the corresponding MI representation of that value.

***AUTL**	Authorization list authority. If this value is specified, no other values can specified. This authority value is only valid for *PUBLIC authority on an object secured by an authorization list.
***EXCLUDE**	Exclude authority. If this value is specified, no other values can be specified.
***ALL**	All authority.
***CHANGE**	Change authority.
***USE**	Use authority.
***OBJOPR**	Object operational authority.
***OBJMGT**	Object management authority.
***OBJEXIST**	Object existence authority.
***READ**	Read authority.
***ADD**	Add authority.
***UPD**	Update authority.
***DLT**	Delete authority.
***AUTLMGT**	Authorization list management authority.

Number of authorities
INPUT; BINARY(4)
The number of authority special values specified in the authority special value parameter. You can specify 1 through 8.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

| CPF22FA E Authority value &1 not valid.
| CPF22FB E Must specify *EXCLUDE or *AUTL as only authority value.
| CPF22F7 E Number of authorities must be between 1 and &1.
| CPF3CF1 E Error code parameter not valid.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Get Profile Handle (QSYGETPH) API

Parameters

Required Parameter Group:

1	User ID	Input	Char(10)
2	Password	Input	Char(10)
3	Profile handle	Output	Char(12)

Optional Parameter:

4	Error code	I/O	Char(*)

The Get Profile Handle (QSYGETPH) API validates user IDs and passwords and creates a profile handle, a type of encrypted pointer, for use in jobs that run under more than one user profile. The profile handle is temporary; you can use it only in the job that created it. In addition, you should use the QSYGETPH API and the Set Profile (QWTSETP) API only to change profiles in server-type jobs. To verify a user's password for other purposes, use the Check Password (CHKPWD) command.

The QSYGETPH API follows this process:

- Verifies that the user ID and password are correct. Incorrect passwords and special cases are handled as follows:
  - On level 10 systems, only the user ID is validated because no passwords are required.
  - If the password is not correct, the incorrect password count is increased. (The QMAXSIGN system value contains the maximum number of incorrect attempts to sign on.) If the QMAXSGNACN system value is set to disable the user profile, repeated attempts to validate an incorrect password disable the user ID. This keeps applications from methodically determining user passwords.
  - If the user ID is *CURRENT, the QSYGETPH API does not verify the password.
  - If the password is *NOPWD, the user requesting the profile handle must have *USE authority to the user profile.
- If the user profile is disabled, the password is expired, or the user's password is *NONE, the call ends without generating a profile handle.
- Generates the profile handle, a 12-character random string designating the user's authorities. This string, not the user's password, supplies the Set Profile (QWTSETP) and the Release Profile Handle (QSYRLSPH) APIs. A single job can create up to 65 534 profile handles; after that, the space to store them is full. Message CPF22E6 is sent to the application, and QSYGETPH stops generating profile handles.

  Be sure to keep track of the profile handles created in the calling application. If the application calls QSYGETPH twice with the same user profile and password, QSYGETPH returns two different profile handles. Either handle can be used, but generating and using just one is more efficient.
- Updates the last-used date for the user and group profiles.
- Resets the signon attempts not valid count to zero.
- If security-related events are being audited, adds an entry to the QAUDJRN audit journal to indicate that a profile handle is created.

Authorities and Locks

If the password is *NOPWD, the user requesting the profile handle needs *USE authority to the profile.

Required Parameter Group

User ID

INPUT; CHAR(10)

The user ID of the profile for which the handle is being created.

You can specify the following special value:

***CURRENT** The handle for the current user profile is generated without validating the password.

Password

INPUT; CHAR(10)

The password for the user ID. On security level 10 systems, the password is not required.

You can specify the following special value:

***NOPWD** If the user currently running has *USE authority to the profile specified in the user ID parameter, no passwords are validated.

The QSYGETPH API does not create a handle for a disabled user profile, one with a password of *NONE, or one with an expired password.

Profile handle

OUTPUT; CHAR(12)

A unique string or handle designating the user profile to use as input to other routines. The handle is temporary; you can use it only in the job that created it.

No profile handle is created if the user profile is disabled, the password has expired, or the password is *NONE.

Optional Parameter

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF22E2 E Password not correct for user profile &1.
CPF22E3 E User profile &1 is disabled.

| CPF22E4 E Password for user profile &1 has expired.
| CPF22E5 E No password associated with user profile &1.
| CPF22E6 E Maximum number of profile handles has been generated.
| CPF22E9 E *USE authority to user profile &1 required.
| CPF2203 E User profile &1 not correct.
| CPF2204 E User profile &1 not found.
| CPF2213 E Not able to allocate user profile &1.
| CPF2225 E Not able to allocate internal system object.
| CPF3CF1 E Error code parameter not valid.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Authorized Users (QSYLAUTU) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Error code	I/O	Char(*)

The List Authorized Users (QSYLAUTU) API puts a list of authorized system users into a user space.

This API provides information similar to the Display Authorized Users (DSPAUTUSR) command.

Authorities and Locks

User Space Authority
*CHANGE

Authority to Library Containing User Space
*USE

Authority to User Profiles in List of Authorized Users
*READ, only those profiles that you have *READ authority to are returned.

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space that the list of authorized users is returned to. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is searched for the user space. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is searched for the user space.

Format name
INPUT; CHAR(8)
The name of the format used to list the authorized users.

You can specify these formats:

AUTU0100 Each entry contains the user name and group name. For a detailed description of this format, see "AUTU0100 Format."
AUTU0200 Each entry contains the same information as AUTU0100 plus the text description for the user. For a detailed description of this format, see "AUTU0200 Format."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions."

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name

AUTU0100 Format

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User name
10	0A	CHAR(10)	Group name

AUTU0200 Format

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User name
10	0A	CHAR(10)	Group name
20	14	CHAR(50)	Text name

Field Descriptions

Format name. The name of the format used to list authorized users.

Group name. The name of the user's group profile. If the user does not have a group profile, this field contains *NONE.

User space library name. The name of the library containing the user space.

Text name. The text description for the authorized user.

User name. The name of the authorized user.

User space name. The name of the user space used to return the list of authorized users on the system.

Error Messages

| CPF2225 E Not able to allocate internal system object.
| CPF3CAA E List is too large for user space &1.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9838 E User profile storage limit exceeded.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Objects Secured by Authorization List (QSYLATLO) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Authorization list	Input	Char(10)
4	Error code	I/O	Char(*)

The List Objects Secured by Authorization List (QSYLATLO) API puts a list of objects secured by an authorization list into a user space.

This API provides information similar to the Display Authorization List Objects (DSPAUTLOBJ) command.

Authorities and Locks

User Space Authority
*CHANGE

Authority to Library Containing User Space
*USE

Authorization List Authority
Must not be *EXCLUDE authority

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space where the list of objects secured by the authorization list is returned to. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the user space. If there is no current library, QGPL (general purpose library)
***LIBL** The library list is used to locate the user space.

Format name
INPUT; CHAR(8)
The name of the format used to list objects secured by the authorization list.

You can specify these formats:

ATLO0100 Each entry contains the object name, library, type, and authority holder indicator. For a detailed description of this format, see "ATLO0100 Format" on page 53-9.
ATLO0200 Each entry contains the same information as ATLO0100 plus the object owner, attribute, and text. For a detailed description of this format, see "ATLO0200 Format" on page 53-9.

Authorization list
INPUT; CHAR(10)
The name of the authorization list for which the secured objects are returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions."

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	Authorization list

Header Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Authorization list

Offset		Type	Field
Dec	Hex		
10	0A	CHAR(10)	User space library name
20	14	CHAR(10)	Owner

ATLO0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder

ATLO0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder
31	1F	CHAR(10)	Owner
41	29	CHAR(10)	Attribute
51	33	CHAR(50)	Text description

Field Descriptions

Attribute. The attribute of the secured object.

Authority holder. Whether the object is an authority holder. If the object is an authority holder, this field is Y. If not, this field is N.

Authorization list. The name of the authorization list for which the list of objects is returned.

Format name. The name of the format used to list objects secured by the authorization list.

Library name. The name of the library containing the user space, object, or authorization list.

Object name. The name of the object secured by the authorization list.

Object type. The type of secured object.

Owner. The name of the owner of the authorization list or object.

Text description. The descriptive text for the secured object.

User space. The user space used to return the list of objects secured by the authorization list.

Error Messages

| CPF22AF E Not authorized to authorization list &1.
| CPF2283 E Authorization list &1 does not exist.
| CPF2289 E Unable to allocate authorization list &1.
| CPF3CAA E List is too large for user space &1.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library
| list.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9838 E User profile storage limit exceeded.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

List Objects That Adopt Owner Authority (QSYLOBJP) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	User	Input	Char(10)
		name	
4	Object type	Input	Char(10)
5	Continuation handle	Input	Char(20)
6	Error code	I/O	Char(*)

The List Objects That Adopt Owner Authority (QSYLOBJP) API puts a list of objects that adopt an object owner's authority into a user space.

This API provides information similar to that provided by the Display Program Adopt (DSPPGMADP) command.

Authorities and Locks

User Space Authority
*CHANGE

Authority to Library Containing User Space
*USE

User Profile Authority
*OBJMGT

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space to which the list of objects that adopt a user's authority is returned. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the user space. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the user space.

Format name
INPUT; CHAR(8)
The name of the format that returns information on the objects that adopt a user's authority.

You can specify these formats:

OBJP0100 Each entry contains the object name, library, type, and object in use indicator. For a detailed description of this format, see "OBJP0100 Format."
OBJP0200 Each entry contains the same information as format OBJP0100 plus the object attribute and descriptive text. For a detailed description of this format, see "OBJP0200 Format."

User name
INPUT; CHAR(10)
The user name for which the list of objects that adopt the user's authority is returned. You can specify the following special value:

***CURRENT** The list of objects that adopt the authority of the user currently running is returned. If *CURRENT is used, the name of the current user is returned in the list header section of the user space.

Object type
INPUT; CHAR(10)
The type of object for which the list of objects that adopt the user's authority is returned. You can specify only the following special values:

***ALL** Return entries for all object types that adopt authority.
***PGM** Return entries for programs that adopt authority.
***SQLPKG** Return entries for SQL packages that adopt authority.
***SRVPGM** Return entries for service programs that adopt authority.

Continuation handle
INPUT; CHAR(20)
The handle used to continue from a previous call to this API that resulted in partially complete information. You can determine if a previous call resulted in partially complete information by checking the Information Status variable in the generic user space header following the API call.

If the API is not attempting to continue from a previous call, this parameter must be set to blanks. Otherwise, a valid continuation value must be supplied. The value may be obtained from the list header section of the user space used in the previous call. When continuing, the first entry in the returned list is the entry that immediately follows the last entry returned in the previous call.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions" on page 53-11.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	User space library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	User name
38	26	CHAR(10)	Object type
48	30	CHAR(20)	Continuation handle

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User name
10	0A	CHAR(20)	Continuation handle

OBJP0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
21	15	CHAR(1)	Object in use

OBJP0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
21	15	CHAR(1)	Object in use
31	1F	CHAR(10)	Attribute
41	29	CHAR(50)	Text description

Field Descriptions

Attribute. The object attribute.

Continuation handle (header section). A continuation point for the API. This value is set based on the contents of the Information Status variable in the generic header for the user space. The following situations can occur:

- Information status–C. The information returned in the user space is valid and complete. No continuation is necessary and the continuation handle is set to blanks.
- Information status–P. The information returned in the user space is valid but incomplete. The user may call the API again, starting where the last call left off. The continuation handle contains a value which may be supplied as an input parameter in later calls.
- Information status–I. The information returned in the user space is not valid and incomplete. The content of the continuation handle is unpredictable.

Continuation handle (input section). Used to continue from a previous call to this API which resulted in partially complete information.

Format name. The name of the format used to return information on the objects that adopt authority.

Library name. The name of the library containing the user space or object.

Object name. The name of the object that adopts the user's authority.

Object in use. Whether the object is in use when the API tries to access it. If the object is in use, the API is not able to determine if the object adopts the user's authority. If the object is in use, this field is Y. If not, this field is N.

Object type.

- Input Section: The type of object for which the list of objects adopting the user's authority is returned.
- List Section: The type of object which adopts the user's authority.

Text description. The text description of the object.

User name. The name of the user for which the list of objects that adopt the user's authority is returned.

User space name. The name of the user space to which the list of objects that adopt the users authority is returned.

Error Messages

| CPF22FD E Continuation handle not valid for API &1.
| CPF2204 E User profile &1 not found.
| CPF2213 E Not able to allocate user profile &1.
| CPF2217 E Not authorized to user profile &1.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF3C31 E Object type &1 is not valid.
| CPF811A E User space &4 in &9 damaged.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &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.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Objects User Is Authorized To or Owns (QSYLOBJA) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	User name	Input	Char(10)
4	Object type	Input	Char(10)
5	Returned objects	Input	Char(10)
6	Continuation handle	Input	Char(20)
7	Error code	I/O	Char(*)

The List Objects a User is Authorized To or Owns (QSYLOBJA) API puts a list of objects a user is authorized to or owns into a user space. The list of authorized objects only includes objects the user is specifically authorized to. The list does not include objects the user is authorized to because:

- The user is part of a group that is authorized
- The user can access the object using the public authority
- The object is secured with an authorization list the user is authorized to
- The user can access the object using adopted authority

This API provides information similar to that provided by the Display User Profile (DSPUSRPRF) command when specifying *OBJAUT or *OBJOWN for the type parameter.

Authorities and Locks

User Space Authority
*CHANGE

Authority to Library Containing User Space
*USE

User Profile Authority
*READ

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space used to return the list of objects a user is authorized to or owns. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the user space. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the user space.

Format name
INPUT; CHAR(8)
The name of the format used to list objects the owner is authorized to and/or owns.

You can specify these formats:

OBJA0100 Each entry contains the object name, library, type, authority holder indicator and ownership indicator. For a detailed description of this format, see "OBJA0100 Format" on page 53-13.
OBJA0200 Each entry contains the same information as format OBJA0100 plus the authority values. For a detailed description of this format, see "OBJA0200 Format" on page 53-13.
OBJA0300 Each entry contains the same information as format OBJA0200 plus the object attribute and descriptive text. For a detailed description of this format, see "OBJA0300 Format" on page 53-13.

User name
INPUT; CHAR(10)
The user name for which the list of authorized or owned objects is being returned. You can specify the following special value:

***CURRENT** The list of objects the user currently signed on is authorized to or owns is returned. If *CURRENT is used, the name of the current user is returned in the list header section of the user space.

Object type
INPUT; CHAR(10)
The type of object the list of authorized or owned objects is returned for. You can specify the following special value:

***ALL** Return entries of all object types.

Returned objects
INPUT; CHAR(10)
The objects that are returned. You can specify the following special values:

***OBJAUT** The list of objects the user is authorized to is returned.
***OBJOWN** The list of objects the user owns is returned.
***BOTH** The list of objects the user is authorized to and owns is returned. The list of owned objects precedes the list of authorized objects.

Continuation handle
INPUT; CHAR(20)
The handle used to continue from a previous call to this API that resulted in partially complete information. You can determine if a previous call resulted in partially complete information by checking the Information Status variable in the generic user space header following the API call.

If the API is not attempting to continue from a previous call, this parameter must be set to blanks. Otherwise, a valid continuation value must be supplied. The value may be obtained from the list header section of the user space used in the previous call. When continuing, the first entry in the returned list is the entry that immediately follows the last entry returned in the previous call.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions" on page 53-13.

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	Library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	User name
38	26	CHAR(10)	Object type
48	30	CHAR(10)	Returned objects
58	3A	CHAR(20)	Continuation handle

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User name
10	0A	CHAR(20)	Continuation handle

OBJA0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder
31	1F	CHAR(1)	Ownership

OBJA0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder
31	1F	CHAR(1)	Ownership
32	20	CHAR(10)	Authority value
42	2A	CHAR(1)	Authorization list management
43	2B	CHAR(1)	Object operational
44	2C	CHAR(1)	Object management
45	2D	CHAR(1)	Object existence
46	2E	CHAR(1)	Data read
47	2F	CHAR(1)	Data add
48	30	CHAR(1)	Data update
49	31	CHAR(1)	Data delete

OBJA0300 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Authority holder
31	1F	CHAR(1)	Ownership
32	20	CHAR(10)	Authority value
42	2A	CHAR(1)	Authorization list management
43	2B	CHAR(1)	Object operational
44	2C	CHAR(1)	Object management
45	2D	CHAR(1)	Object existence
46	2E	CHAR(1)	Data read
47	2F	CHAR(1)	Data add
48	30	CHAR(1)	Data update
49	31	CHAR(1)	Data delete
50	32	CHAR(10)	Attribute
60	3C	CHAR(50)	Text description

Field Descriptions

Attribute. The object's attribute.

Authority holder. Whether the object is an authority holder. If the object is an authority holder, this field is Y. If not, this field is N.

Authority value. The special value indicating the user's authority to the object. This field contains one of the following values:

**ALL* The user has all object (operational, management, and existence) and data (read, add, update, and delete) authorities to the object.

**CHANGE* The user has object operational and all data authorities to the object.

**USE* The user has object operational and data read authorities to the object.

**EXCLUDE* The user has none of the object or data authorities to the object, or authorization list management authority.

USER DEF The user has some combination of object and data authorities that do not relate to a special value. The individual authorities for the user should be checked to determine what authority the user has to the object. This value is returned if the user owns an object and all authority for the user to the object has been removed. If this happens, all individual authority fields are set to N.

Authorization list management. Whether the user has authorization list management authority to the object. If the user has the authority, this field is Y. If not, this field is N. This field is only valid if the object type is *AUTL.

Continuation handle (header section). A continuation point for the API. This value is set based on the contents of the Information Status variable in the generic header for the user space. The following situations can occur:

- Information status–C. The information returned in the user space is valid and complete. No continuation is necessary and the continuation handle is set to blanks.
- Information status–P. The information returned in the user space is valid but incomplete. The user may call the API again, picking up where the last call ended. The continuation handle contains a value, that may be supplied as an input parameter in later calls.
- Information status–I. The information returned in the user space is not valid or complete. The contents of the continuation handle are unpredictable.

Continuation handle (input section). The handle used to continue from a previous call to this API that resulted in partially complete information.

Data add. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data delete. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data read. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data update. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Format name. The name of the format used to list objects the user is authorized to or owns.

Library name. The name of the library containing the user space or object.

Object name. The name of the object the user is authorized to or owns.

Object existence. Whether the user has object existence authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object management. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object operational. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object type.

Input Section
The type of object for which the list of authorized or owned objects is returned.

List Section
The type of object the user is authorized to or owns.

Ownership. Whether the user owns the object. If the user owns the object, this field is Y. If not, this field is N.

Returned objects. The objects that are returned.

Text description. The text description of the object.

User name. The user name for which the list of authorized or owned objects is returned.

User space name. The name of the user space used to return the list of objects the user is authorized to or owns.

Error Messages

| CPF22FC E Value &1 not valid when specifying objects to be returned by API &2.
| CPF22FD E Continuation handle not valid for API &1.
| CPF2204 E User profile &1 not found.
| CPF2213 E Not able to allocate user profile &1.
| CPF2217 E Not authorized to user profile &1.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF3C31 E Object type &1 is not valid.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Users Authorized to Object (QSYLUSRA) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified object name	Input	Char(20)
4	Object type	Input	Char(10)
5	Error code	I/O	Char(*)

The List Users Authorized to Object (QSYLUSRA) API puts a list of users privately authorized to an object, including an authorization list, into a user space. The information returned is the authority as it exists for the object. Any authority the process has to the object through its group or adopted authority is not included. *PUBLIC authority to the object is also returned in the first list entry of the user space.

This API provides information similar to that provided by the Display Authorization List (DSPAUTL) command or the Display Object Authority (DSPOBJAUT) command.

Authorities and Locks

User Space Authority
*CHANGE
Authority to Library Containing User Space
*USE
Specified Object or Authorization List Authority
*OBJMGT

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The name of the existing user space used to return the list of authorized users to the object. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the user space. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the user space.

Format name
INPUT; CHAR(8)
The name of the format used to list authorized users.

You can specify this format:

USRA0100 Each entry contains the user name and authority values. For a detailed description of this format, see "USRA0100 Format."

Qualified object name
INPUT; CHAR(20)
The name of the object for which the list of authorized users is returned. The first 10 characters specify the object name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the object. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the object.

Object type
INPUT; CHAR(10)
The type of object for which the list of authorized users is returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see "Field Descriptions."

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name
10	0A	CHAR(10)	Library name
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	Object name
38	26	CHAR(10)	Library name
48	30	CHAR(10)	Object type

Header Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(10)	Owner name
40	28	CHAR(10)	Authorization list

USRA0100 Format

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User name
10	0A	CHAR(10)	Authority value
20	14	CHAR(1)	Authorization list management
21	15	CHAR(1)	Object operational
22	16	CHAR(1)	Object management
23	17	CHAR(1)	Object existence
24	18	CHAR(1)	Data read
25	19	CHAR(1)	Data add
26	1A	CHAR(1)	Data update
27	1B	CHAR(1)	Data delete

Field Descriptions

Authority value. The user's authority to the object. This field contains one of the following values:

**ALL* The user has all object (operational, management, and existence) and data (read, add, update, and delete) authorities to the object.

*CHANGE The user has object operational and all data authorities to the object.

*USE The user has object operational and data read authorities to the object.

*EXCLUDE The user has none of the object or data authorities to the object, or authorization list management authority to the authorization list.

*AUTL The public authority for the object comes from the public authority on the authorization list securing the object. This value can only be returned if there is an authorization list securing the object and the authorized user is *PUBLIC.

USER DEF The user has some combination of object and data authorities that do not relate to a special value. The individual authorities for the user should be checked to determine what authority the user has to the object.

Authorization list. The name of the authorization list securing the object. If there is no authorization list securing the object, this field is *NONE.

Authorization list management. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N. This field is only valid if the object type is *AUTL.

Data add. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data delete. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data read. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data update. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Format name. The name of the format used to list users authorized to the object.

User space library name. The name of the library containing the user space or object.

Object name. The name of the object for which the list of authorized users is returned.

Object existence. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object management. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object operational. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object type. The type of object for which the list of authorized users is returned.

Owner. The name of the owner of the object. If all authority for the owner is removed, no list entry is returned for the owner.

User name. The name of the user authorized to the object. This field can contain the following special value:

*PUBLIC Public authority (authority used by users not privately authorized) to the object. This is the first entry in the list data section.

User space name. The name of the user space used to return the list of users authorized to the object.

Error Messages

| CPF3CAA E List is too large for user space &1.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF3C31 E Object type &1 is not valid.
| CPF811A E User space &4 in &9 damaged.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &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.

Release Profile Handle (QSYRLSPH) API

Parameters

Required Parameter:

1	Profile handle	Input	Char(12)

Optional Parameter:

2	Error code	I/O	Char(*)

The Release Profile Handle (QSYRLSPH) API validates a given profile handle and then deletes it. To use the user profile represented by the deleted profile handle in the job again, you must call the Get Profile Handle (QSYGETPH) API to generate a new profile handle for the user profile.

Your application must perform any needed cleanup work like closing files and deallocating objects. The QSYRLSPH API only releases the profile handle; it does not perform any cleanup in the job.

Required Parameter

Profile handle
INPUT; CHAR(12)
The profile handle to be released.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

| CPF3CF1 E Error code parameter not valid.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve User Authority to Object (QSYRUSRA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Format name	Input	Char(8)
4	User name	Input	Char(10)
5	Qualified object name	Input	Char(20)
6	Object type	Input	Char(10)
7	Error code	I/O	Char(*)

The Retrieve User Authority to Object (QSYRUSRA) API returns a specific user's authority for an object to the call.

Authorities and Locks

The following authorities are required for the user calling this API, unless the user specified is *CURRENT, the caller owns the object, or the object is an authorization list:

- *OBJMGT authority to the object specified.
- *READ authority to the user profile specified (unless *PUBLIC is specified).

If previous programs in the program stack adopt their owner's authority, the adopted authority for the current program is the accumulated adopted authority from all other programs in the program stack that adopt authority. Adopted authority is only valid when the user specified is *CURRENT.

When retrieving authority to logical files, no data authorities (read, add update, or delete) are returned. Logical files do not have data authorities associated with them; the data authorities are associated with the base physical file.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable used to return the user's authority to the object. This variable must be at least 8 bytes long.

Receiver variable length
INPUT; BINARY(4)
The length of the receiver variable. The variable must be at least 8 bytes long.

Format name
INPUT; CHAR(8)
The name of the format used to return the authority information. You can specify the following special value:

USRA0100 All authority information is returned. For a detailed description of this format, see "USRA0100 Format" on page 53-18.

User name
INPUT; CHAR(10)
The name of the user whose object authority is returned. You can specify the following special values:

***CURRENT** The authority of the user currently running to the specified object is returned.
***PUBLIC** The public authority for the object is returned.

Qualified object name
INPUT; CHAR(20)
The name of the object whose authority is returned. The first 10 characters specify the object name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the object. If there is no current library, QGPL (general purpose library) is used.
***LIBL** The library list is used to locate the object.

Object type
INPUT; CHAR(10)
The type of object for which authority information is returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Receiver Variable Description

The following tables describe the order and format of the data returned in the receiver variable. For detailed descriptions of the fields in the tables, see "Field Descriptions" on page 53-18.

USRA0100 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Object authority
18	12	CHAR(1)	Authorization list management
19	13	CHAR(1)	Object operational
20	14	CHAR(1)	Object management
21	15	CHAR(1)	Object existence
22	16	CHAR(1)	Data read
23	17	CHAR(1)	Data add
24	18	CHAR(1)	Data update
25	19	CHAR(1)	Data delete
26	1A	CHAR(10)	Authorization list
36	24	CHAR(2)	Authority source
38	26	CHAR(1)	Some adopted authority
39	27	CHAR(10)	Adopted object authority
49	31	CHAR(1)	Adopted authorization list management
50	32	CHAR(1)	Adopted object operational
51	33	CHAR(1)	Adopted object management
52	34	CHAR(1)	Adopted object existence
53	35	CHAR(1)	Adopted data read
54	36	CHAR(1)	Adopted data add
55	37	CHAR(1)	Adopted data update
56	38	CHAR(1)	Adopted data delete

Field Descriptions

Adopted authorization list management. Whether the user has adopted this authority to the object. If the user adopted the authority, this field is Y. If not, this field is N.

Adopted data add. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted data delete. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted data read. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted data update. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted object authority. The user's adopted authority to the object. This field is only valid if some of the user's authority is adopted. If the user does not adopt authority, this field and the other adopted authority fields will be blank. If the user adopts authority, this field contains one of the following values:

**ALL* The user adopted all object (operational, management, and existence) and data (read, add, update, and delete) authorities to the object.

**CHANGE* The user adopted object operational and all data authorities to the object.

**USE* The user adopted object operational and data read authorities to the object.

USER DEF The user adopted some combination of object and data authorities that do not relate to a special value. The individual authorities for the user should be checked to determine what authority the user has adopted to the object.

Adopted object existence. Whether the user adopted this authority to the object. If the user adopted the authority, this field is Y. If not, this field is N.

Adopted object management. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Adopted object operational. Whether the user has adopted this authority to the object. If the user has adopted the authority, this field is Y. If not, this field is N.

Authority source. Indicates where the authority that the user has to the object initially came from. The authority may be a combination of authority from this source plus adopted authority. This field contains one of the following special values:

UA The user has *ALLOBJ special authority.

UO The user is privately authorized to the object.

UL The user is privately authorized to the authorization list securing the object.

GA The user's group has *ALLOBJ special authority.

GO The user's group is privately authorized to the object.

GL The user's group is privately authorized to the authorization list securing the object.

PO The user accesses the object through the public authority.

PL The user accesses the object through the public authority on the authorization list securing the object.

AD All of the authority that the user has comes from adopted authority. This value is only returned if the user is *CURRENT.

Authorization list. The name of the authorization list securing the object. This field can contain one of the following special values:

**NONE* There is no authorization list securing the object.
**DAMAGED* The authorization list securing the object is damaged.

Authorization list management. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Bytes available. The number of bytes of data available to be returned to the user. If all data is returned, this is the same as the number of bytes returned. If the receiver variable was not big enough to contain all of the data, this is the number of bytes that can be returned.

Bytes returned. The number of bytes of data returned to the user. This is the lesser of the number of bytes available to be returned or the length of the receiver variable.

Data add. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data delete. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data read. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Data update. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object authority. The special value indicating the user's authority to the object. This is the user's total authority to the object, including adopted authority (if the user is *CURRENT). This is one of the following values:

**ALL* The user has all object (operational, management, and existence) and data (read, add, update, and delete) authorities to the object.
**CHANGE* The user has object operational and all data authorities to the object.
**USE* The user has object operational and data read authorities to the object.
**EXCLUDE* The user has none of the object or data authorities to the object, or authorization list management authority.
USER DEF The user has some combination of object and data authorities that do not relate to a special value. The individual authorities for the user should be checked to determine what authority the user has to the object.

Object existence. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object management. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Object operational. Whether the user has this authority to the object. If the user has the authority, this field is Y. If not, this field is N.

Some adopted authority. Whether some of the authority that the user has to the object comes from adopted authority. If some of the authority is adopted, this field is Y. If not, this field is N. This field can only contain Y if the user is *CURRENT.

Error Messages

| CPF2203 E User profile &1 not correct.
| CPF2225 E Not able to allocate internal system object.
| CPF3CF1 E Error code parameter not valid.
| CPF3C19 E Error occurred with receiver variable specified.
| CPF3C21 E Format name &1 is not valid.
| CPF3C24 E Length of the receiver variable is not valid.
| CPF3C31 E Object type &1 is not valid.
| CPF8122 E &8 damage on library &4.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9811 E Program &1 in library &2 not found.
| CPF9812 E File &1 in library &2 not found.
| CPF9814 E Device &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve User Information (QSYRUSRI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Format name	Input	Char(8)
4	User name	Input	Char(10)
5	Error code	I/O	Char(*)

The Retrieve User Information (QSYRUSRI) API provides information about a user profile. This API provides information similar to the Retrieve User Profile (RTVUSRPRF) command or the Display User Profile (DSPUSRPRF) command when *BASIC is specified for the type parameter.

Authorities and Locks

User Profile Authority

*READ

Required Parameter Group

Receiver variable

OUTPUT; CHAR(*)

The variable used to return the information about the user. This variable must be at least 8 bytes long.

Receiver variable length

INPUT; BINARY(4)

The length of the receiver variable. The variable must be 8 bytes long.

Format name

INPUT; CHAR(8)

The name of the format used to return information about the user.

You can specify these formats:

USRI0100 Sign-on and password information is returned. The password itself is not returned. For a detailed description of this format, see "USRI0100 Format."

USRI0200 Authority information is returned. For a detailed description of this format, see "USRI0200 Format."

USRI0300 All user information is returned. For a detailed description of this format, see "USRI0300 Format."

User name

INPUT; CHAR(10)

The user name for which information is returned. You can specify the following special value:

***CURRENT** The information for the user currently running is returned.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Receiver Variable Description

The following tables describe the order and format of the data returned in the receiver variable. For detailed descriptions of the fields in the tables, see "Field Descriptions" on page 53-21.

USRI0100 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	User profile name
18	12	CHAR(13)	Previous sign-on date and time
31	1F	CHAR(1)	Reserved
32	20	BINARY(4)	Sign-on attempts not valid
36	24	CHAR(10)	Status
46	2E	CHAR(8)	Password change date
54	36	CHAR(1)	No password indicator
55	37	CHAR(1)	Reserved
56	38	BINARY(4)	Password expiration interval
60	3C	CHAR(8)	Date password expires
68	44	BINARY(4)	Days until password expires
72	48	CHAR(1)	Set password to expire
73	49	CHAR(10)	Display sign-on information

USRI0200 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	User profile name
18	12	CHAR(10)	User class name
28	1C	CHAR(15)	Special authorities
43	2B	CHAR(10)	Group profile name
53	35	CHAR(10)	Owner
63	3F	CHAR(10)	Group authority name
73	49	CHAR(10)	Limit capabilities

USRI0300 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	User profile name
18	12	CHAR(13)	Previous sign-on date and time
31	1F	CHAR(1)	Reserved
32	20	BINARY(4)	Sign-on attempts not valid
36	24	CHAR(10)	Status
46	2E	CHAR(8)	Password change date
54	36	CHAR(1)	No password indicator
55	37	CHAR(1)	Reserved

Offset		Type	Field
Dec	Hex		
56	38	BINARY(4)	Password expiration interval
60	3C	CHAR(8)	Date password expires
68	44	BINARY(4)	Days until password expires
72	48	CHAR(1)	Set password to expire
73	49	CHAR(10)	User class name
83	53	CHAR(15)	Special authorities
98	62	CHAR(10)	Group profile name
108	6C	CHAR(10)	Owner
118	76	CHAR(10)	Group authority
128	80	CHAR(10)	Assistance level
138	8A	CHAR(10)	Current library name
148	94	CHAR(10)	Initial menu name
158	9E	CHAR(10)	Initial menu library name
168	A8	CHAR(10)	Initial program name
178	B2	CHAR(10)	Initial program library name
188	BC	CHAR(10)	Limit capabilities
198	C6	CHAR(50)	Text description
248	F8	CHAR(10)	Display sign-on information
258	102	CHAR(10)	Limit device sessions
268	10C	CHAR(10)	Keyboard buffering
278	116	CHAR(2)	Reserved
280	118	BINARY(4)	Maximum allowed storage
284	11C	BINARY(4)	Storage used
288	120	CHAR(1)	Highest scheduling priority
289	121	CHAR(10)	Job description name
299	123	CHAR(10)	Job description name library
309	134	CHAR(15)	Accounting code
324	144	CHAR(10)	Message queue name
334	14E	CHAR(10)	Message queue library name
344	158	CHAR(10)	Message queue delivery method
354	162	CHAR(2)	Reserved
356	164	BINARY(4)	Message queue severity
360	168	CHAR(10)	Output queue
370	172	CHAR(10)	Output queue library
380	17C	CHAR(10)	Print device
390	186	CHAR(10)	Special environment
400	190	CHAR(10)	Attention-key-handling program name
410	19A	CHAR(10)	Attention-key-handling program library
420	1A4	CHAR(10)	Language ID
430	1AE	CHAR(10)	Country ID
440	1B8	BINARY(4)	Character code set ID
444	1BC	CHAR(36)	User options
480	1E0	CHAR(10)	Sort sequence table name
490	1EA	CHAR(10)	Sort sequence table library name
500	1F4	CHAR(10)	Object audit value
510	1FE	CHAR(64)	User action audit level

Field Descriptions

Accounting code. The accounting code associated with this user. If the user does not have an accounting code, this field is blank.

Assistance level. The user interface the user will use. The field contains one of the following values:

**SYSVAL* The system value QASTLVL determines which user interface the user is using.
**BASIC* The user uses the Operational Assist user interface.
**INTERMED* The user uses the system user interface.
**ADVANCED* The user uses the expert system user interface.

Attention-key-handling program library name. The name of the library where the program is located. This field can contain the special value of *LIBL. If the program name is a special value, this field is blank.

Attention-key-handling program name. The Attention-key-handling program for this user. This field contains the following fields:

- CHAR(10): The name of the Attention-key-handling program. This field may contain one of the following special values:
  **SYSVAL* The system value, QATNPGM, is used to determine the user's Attention-key-handling program.
  **NONE* No Attention-key-handling program is used by this user.
  **ASSIST* The Operational Assistant Attention-Key-Handling API (QEZMAIN) is used by this user.

Bytes available. The number of bytes of data available to be returned to the user. If all data is returned, this is the same as the number of bytes returned. If the receiver variable was not big enough to contain all of the data, this is the number of bytes that can be returned.

Bytes returned. The number of bytes of data returned to the user. This is the lesser of the number of bytes available to be returned or the length of the receiver variable.

Character code set ID. The character code set ID to be used by the system for this user. This field can contain the following special values:

-2 The system value QCCSID is used to determine the user's character code set ID.

Country ID. The country ID used by the system for this user. This field can contain the following special value:

**SYSVAL* The system value QCNTRYID is used to determine the user's country ID.

Current library name. This field contains the name of the user's current library. If the user does not have a current library, this field is *CRTDFT.

Date password expires. The date the user's password expires, in *DTS (date-time stamp) format. If the user's password will not expire (password expiration interval of *NOMAX) or the user's password is set to expire, then this field is blank.

Days until password expires. The number of days until the password will expire. This field contains one of the following values:

0 The password is expired.
1–7 The number of days until the password expires.
-1 The password will not expire in the next 7 days.

Display sign-on information. Whether the sign-on information display is shown when the user signs on. The field contains one of the following values:

**SYSVAL* The system value QDSPSGNINF determines if the sign-on information display is shown when the user signs on.
**YES* The sign-on information display is shown when the user signs on.
**NO* The sign-on information display is not shown when the user signs on.

Group authority. The authority the user's group profile has to objects the user creates. The field contains one of the following values:

**NONE* The group profile has no authority to the objects the user creates. If the user does not have a group profile, the field contains this value.
**ALL* The group profile has all authority to the objects the user creates.
**CHANGE* The group profile has change authority to the objects the user creates.
**USE* The group profile has use authority to the objects the user creates.
**EXCLUDE* The group profile has exclude authority to the objects the user creates.

Group profile name. The name of the group profile. If this user does not have a group profile, this field is *NONE.

Highest scheduling priority. The highest scheduling priority the user is allowed to have for each job submitted to the system. The priority is a value from 0 through 9, with 0 being the highest priority.

Initial menu name. The initial menu for the user. This field contains the following fields:

- CHAR(10): The name of the menu. This field can contain the special value *SIGNOFF.
- CHAR(10): The name of the library where the menu is located. This field can contain the special value of *LIBL. If the menu name is *SIGNOFF, this field is blank.

Initial menu library name. The name of the library that the initial menu is in.

Initial program name. The initial program for the user. This field contains the following fields:

- CHAR(10): The name of the program. If the user does not have an initial program, this field is *NONE.
- CHAR(10): The name of the library where the program is located. This field can contain the special value of *LIBL. If the program name is *NONE, this field is blank.

Initial program library name. The name of the library that the initial program is in.

Job description name. The name of the job description used for jobs that start through subsystem work station entries. This field contains the following fields:

- CHAR(10): The name of the job description.
- CHAR(10): The name of the library where the job description is located or the special value *LIBL.

Job description name library. The name of the library that the job description is in.

Keyboard buffering. This field indicates the keyboard buffering value that is used when a job is initialized for this user. The field contains one of the following values:

**SYSVAL* The system value QKBDBUF determines the keyboard buffering value for this user.
**YES* The type-ahead and attention-key buffering options are both on.
**NO* The type-ahead and attention-key buffering options are not on.
**TYPEAHEAD* The type-ahead option is on, but the attention-key buffering option is not.

Language ID. The language ID used by the system for this user. This field can contain the following special value:

**SYSVAL* The system value QLANGID is used to determine the user's language ID.

Limit capabilities. Whether the user has limited capabilities. The field contains one of the following values:

**PARTIAL* The user cannot change his initial program or current library.

*YES The user cannot change his initial menu, initial program, or current library. The user cannot run commands from the command line.
*NO The user is not limited.

Limit device sessions. Whether the user is limited to one device session. The field contains one of the following values:

*SYSVAL The system value QLMTDEVSSN determines if the user is limited to one device session.
*YES The user is limited to one device session.
*NO The user is not limited to one device session.

Maximum allowed storage. The maximum amount of auxiliary storage (in kilobytes) that can be assigned to store permanent objects owned by the user. If the user does not have a maximum amount of allowed storage, this field contains -1 for *NOMAX.

Message queue name. The name of the message queue that is used by this user. This field contains the following fields:

- CHAR(10): The name of the message queue.
- CHAR(10): The name of the library where the message queue is located or the special value *LIBL.

Message queue library name. The name of the library the message queue is in.

Message queue delivery method. How the messages are delivered to the message queue used by the user. This field contains one of the following special values:

*BREAK The job to which the message queue is assigned is interrupted when a message arrives on the message queue.
*DFT Messages requiring replies are answered with their default reply.
*HOLD The messages are held in the message queue until they are requested by the user or program.
*NOTIFY The job to which the message queue is assigned is notified when a message arrives on the message queue.

Message queue severity. The lowest severity that a message can have and still be delivered to a user in break or notify mode. The severity is a value from 0 through 99.

No password indicator. If *NONE is specified for the password in the user profile, this field contains a Y. If not, this field is N.

Output queue name. The output queue used by this user. This field contains the following fields:

- CHAR(10): The name of the output queue used by the user. This field can contain one of the following special values:
  *WRKSTN The output queue assigned to the user's work station is used.
  *DEV An output queue with the same name as the device specified in the printer device parameter is used by the user.

Object auditing value. The current user's object auditing value. The field contains one of the following values:

*NONE No additional object auditing is done for the current user.
*CHANGE Object changes are audited for the current user if the object's auditing value is *USRPRF.
*ALL Object read and change operations are audited for the current user if the object's auditing value is *USRPRF.

Output queue library name. The name of the library where the output queue is located. This field can contain the special value *LIBL. If the output queue is *WRKSTN or *DEV, this field is blank.

Owner. This field indicates who is to own objects created by this user. The field contains one of the following values:

*USRPRF The user owns any objects the user creates. If the user does not have a group profile, the field contains this value.
*GRPPRF The user's group profile owns any objects the user creates.

Password change date. The date the user's password was last changed, in *DTS (date-time stamp) format.

Password expiration interval. The number of days (from 1 through 366) the user's password can remain active before it must be changed. This field may contain one of the following special values:

0 The system value QPWDEXPITV is used to determine the user's password expiration interval.
-1 The user's password does not expire (*NOMAX).

Previous sign-on date and time. The date and time the user last signed on. The 13 characters are:

- CHAR(1): The century. 0 indicates the twentieth century and 1 indicates the twenty-first century.
- CHAR(6): The date, in YYMMDD (year, month, day) format.
- CHAR(6): The time, in HHMMSS (hours, minutes, seconds) format.

Print device. The printer used to print for this user. This field can contain one of the following special values:

*WRKSTN The printer assigned to the user's work station is used.
*SYSVAL The default system printer specified in the system value QPRTDEV is used.

Reserved. An ignored field.

Set password to expire. Whether the user's password is set to expire, requiring the user to change the password

when signing on. This field contains one of the following values:

Y The user's password is set to expire.
N The user's password is not set to expire.

Sign-on attempts not valid. The number of sign-on attempts that were not valid since the last successful sign-on.

Sort sequence table name. The name of the sort sequence table used for string comparisons. The following possible special values can also be returned:

**HEX* The hexadecimal values of the characters are used to determine the sort sequence.
**LANGIDUNQ* A unique-weight sort table associated with the language specified.
**LANGIDSHR* A shared-weight sort table associated with the language specified.
**SYSVAL* The system value QSRTSEQ.

Sort sequence table library name. The name of the library that is used to locate the sort sequence table. This information is blank if the program does not contain any sort sequence information.

Special authorities. The special authorities the user has. If the user has the special authority, the field is Y. If not, the field is N. This field contains the following fields:

- CHAR(1): All object. Whether the user has all object special authority.
- CHAR(1): Security administrator. Whether the user has security administrator special authority.
- CHAR(1): Job control. Whether the user has job control special authority.
- CHAR(1): Spool control. Whether the user has spool control special authority.
- CHAR(1): Save system. Whether the user has save system special authority.
- CHAR(1): Service. Whether the user has service special authority.
- CHAR(1): Audit. Whether the user has audit special authority.

Special environment. The special environment the user operates in after signing on. This field contains one of the following special values:

**SYSVAL* The system value QSPCENV is used to determine the user's special environment.
**NONE* The user operates in the OS/400 environment.
**S36* The user operates in the System/36 environment.

Status. The status of the user profile. This field contains one of the following values:

**ENABLED* The user profile is enabled; therefore, the user is able to sign on.
**DISABLED* The user profile is disabled; therefore, the user cannot sign on.

Storage used. The amount of auxiliary storage (in kilobytes) occupied by this user's owned objects.

Text description. The descriptive text for the user profile.

User action audit level. The action audit values for this user. If the user has a specific audit value, the field is Y. If not, the field is N. This field contains the following:

- *CMD CHAR(1): The user has the *CMD audit value specified in the user profile.
- *CREATE CHAR(1): The user has the *CREATE audit value specified in the user profile.
- *DELETE CHAR(1): The user has the *DELETE audit value specified in the user profile.
- *JOBDTA CHAR(1): The user has the *JOBDTA audit value specified in the user profile.
- *OBJMGT CHAR(1): The user has the *OBJMGT audit value specified in the user profile.
- *OFCSRV CHAR(1): The user has the *OFCSRV audit value specified in the user profile.
- *PGMADP CHAR(1): The user has the *PGMADP audit value specified in the user profile.
- *SAVRST CHAR(1): The user has the *SAVRST audit value specified in the user profile.
- *SECURITY CHAR(1): The user has the *SECURITY audit value specified in the user profile.
- *SERVICE (CHAR(1): The user has the *SERVICE audit value specified in the user profile.
- *SPLFDTA CHAR(1): The user has the *SPLFDTA audit value specified in the user profile.
- *SYSMGT CHAR(1): The user has the *SYSMGT audit value specified in the user profile.
- Reserved CHAR(52): An ignored field.

User class name. This field contains one of the following special values:

**SECOFR* The user has a class of security officer.
**SECADM* The user has a class of security administrator.
**PGMR* The user has a class of programmer.
**SYSOPR* The user has a class of system operator.
**USER* The user has a class of end user.

User options. The options for users to customize their environment. This field contains the following fields:

- CHAR(1): Show keywords (*CLKWD). Whether the keywords are shown when a CL command is displayed. If the keywords are to be shown, this field is Y. If not, this field is N.
- CHAR(1): Show detailed information (*EXPERT). Whether more detailed information is shown when the user is defining or changing the system using edit or display object authority. This user option is independent of the ASTLVL parameter on the user profile and the ASTLVL parameter available on commands. If the details are to be shown, this field is Y. If not, this field is N.
- CHAR(1): Full screen help (*HLPFULL). Whether UIM online help is to be displayed on a full screen or in a

window. If the full screen is to be shown, this field is Y. If not, this field is N.
- CHAR(1): Show status message (*STSMSG). Whether status messages sent to the user are shown. If the status messages are to be shown, this field is Y. If not, this field is N.
- CHAR(1): Do not show status message (*NOSTSMSG). Whether status messages sent to the user are not shown.
- CHAR(1): Roll key direction change (*ROLLKEY). Whether the opposite action from the system default for roll keys is taken or not. If the opposite action is to be taken, this field is Y. If not, this field is N.
- CHAR(1): Printing complete message (*PRTMSG). Whether a message is sent to the user when a spooled file is printed or not. If a message is to sent to the user, this field is Y. If not, this field is N.
- CHAR(29): Reserved.

User profile name. The name of the user profile for which the information is returned.

Error Messages

| CPF2203 E User profile &1 not correct.
| CPF2225 E Not able to allocate internal system object.
| CPF3CF1 E Error code parameter not valid.
| CPF3C19 E Error occurred with receiver variable specified.
| CPF3C21 E Format name &1 is not valid.
| CPF3C24 E Length of the receiver variable is not valid.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 in &3.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Set Profile (QWTSETP) API

Parameters

Required Parameter:

1	Profile handle	Input	Char(12)

Optional Parameter:

2	Error code	I/O	Char(*)

The Set Profile (QWTSETP) API validates the profile handle, locks the user profile, and changes the job to run under the user and group profile represented by the profile handle. Once the change has been made, any open files and objects allocated by the original profile are accessible to the new profile.

No other attributes associated with the user or group profile are replaced. The qualified job name does not change to reflect the new user profile. However, any object created by the job while running under the new profile are owned by the new profile or its group profile.

If the profile handle is not valid, the QWTSETP API releases all profile handles previously created by the Get Profile Handle (QSYGETPH) API for the job, adds an exception to the job log, and enters a security violation in the QAUDJRN audit journal.

| If you use this API to begin running under a specific profile, any spooled files created are owned by that profile. This is done by putting the file under a QPRTJOB job.

| QPRTJOB

| A QPRTJOB job is the name of a job that files are spooled under when the current job's user name is not the same as the user profile currently running. For example, if you use QWTSETP to set the profile to user xxxx and create a spooled file, the file is spooled under job 999999/xxxx/QPRTJOB. This ensures that user xxxx owns the spooled file and if that user uses the WRKSPLF command, the file is displayed.

| Output Queue Considerations

| The output queue a spooled file is placed in may be different after using this API. For example, if you press the print key to produce a print screen spooled file, before changing profiles, the output queue that file is placed on is determined by the following.

| Before Changing Profile

| • Printer file parameter output queue

| In this case the printer file is QSYSPRT with an output queue value of *JOB.
| • The output queue for your job is normally defaulted to *USRPRF, which is the output queue value in your user profile.
| • The output queue for your user profile may be a specific output queue or the default *WRKSTN.
| • The device configuration for your workstation normally defaults the value for the output queue to *DEV.

| This tells the job to start again at the printer file level and look at the device parameter. The device parameter has similar defaults as the output queue parameter except the last default found is the system value for QPRTDEV. The spooled file is placed on a device output queue.

| If you set the profile to X using this API, when you press the print key to create a spooled file, the file may be placed on a different output queue. The output queue is determined as follows, after setting your job to a different profile.

After Setting Profile

- Printer file parameter output queue

  In this case the printer file is QSYSPRT with an output queue value of *JOB.
- The job output queue is determined by the QPRTJOB job instead of the job currently running. All QPRTJOB jobs have a value of *DEV for output queue, which means to start at the printer file to determine which device output queue is to be used.
- The QSYSPRT printer file parameter for the device is *JOB.

  The job's device is taken from the job that is running, not the QPRTJOB job. A job's device parameter is normally defaulted to *USRPRF. The user profile may contain a device value or specify the default *WRKSTN. The user profile used is the one associated with the job that is running, not the user profile set with QWTSETP. If you use all the defaults, the printer device system value is QPRTDEV.

For more information on QPRTJOB jobs see the *Guide to Programming for Printing*.

Required Parameter

Profile handle
INPUT; CHAR(12)
The profile handle returned by the QSYGETPH API for the user profile to switch the job to.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF22AD E Group profile for user not found.
CPF22E7 E Profile handle is not valid.
CPF2204 E User profile &1 not found.
CPF2213 E Not able to allocate user profile &1.
CPF2217 E Not authorized to user profile &1.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 20. Software Product APIs

Chapter 54. Software Product APIs

This chapter discusses the APIs that let you work with software products and program temporary fixes (PTFs) on your system. These APIs allow you to do the following:

- Work with user-based licensing for a product
- Create and delete product definition and product load objects
- Package one or more product loads for a specified product option
- Retrieve product information about a specific product load
- Create, work with, and retrieve PTFs
- Report a software problem to your AS/400 system and gather data for use in resolving the problem

The software product APIs are:

Add Product License Information (QLZADDLI) adds license information to a product or feature.
Create Product Definition (QSZCRTPD) creates a product definition object.
Create Product Load (QSZCRTPL) creates a product load object.
Create Program Temporary Fix (QPZCRTFX) creates a PTF save file and optionally creates a PTF cover letter.
Delete Product Definition (QSZDLTPD) deletes a product definition object.
Delete Product Load (QSZDLTPL) deletes a single product load object.
Generate Program Temporary Fix Name (QPZGENNM) generates a unique name for PTF save files and cover letters.
Log Program Temporary Fix Information (QPZLOGFX) logs that a PTF has been received on the system and can be displayed or loaded.
Log Software Error (QPDLOGER) reports a software problem and collects data needed for its resolution.
Package Product Option (QSZPKGPO) packages one or more product loads for a specified product option.
Release License (QLZARLS) releases a use of the license for the product.
Request License (QLZAREQ) requests a use of the license for the product.
Retrieve License Information (QLZARTV) retrieves the license information for a software product.
Retrieve Product Information (QSZRTVPR) retrieves information about a specific product load for a software product.
Retrieve Program Temporary Fix Information (QPZRTVFX) retrieves basic information about and prerequisites for a PTF.

The APIs in this chapter are presented in alphabetical order.

Add Product License Information (QLZADDLI) API

Parameters

Required Parameter Group:

1	Product identification	Input	Char(*)
2	Product identification format name	Input	Char(8)
3	License information	Input	Char(*)
4	License information format name	Input	Char(8)
5	Error code	I/O	Char(*)

The Add Product License Information (QLZADDLI) API adds license information to a product or a feature of a product. License information can be added at two times:

- After the product definition object (*PRDDFN) has been created using the Create Product Definition (CRTPRDDFN) command or the Create Product Definition (QSZCRTPD) API
- Either before or after the product options have been packaged using the Package Product Option (PKGPRDOPT) command or the Package Product Option (QSZPKGPO) API

License information must be added before the product is saved with the Save License Program (SAVLICPGM) command.

Authorities and Locks

Product Availability Lock
*SHRRD. The product availability object is in the QUSRSYS library.

Public Authority *EXCLUDE

Required Parameter Group

Product identification
INPUT; CHAR(*)
Information that uniquely identifies the product or feature to which license information will be added. The structure of this information is determined by the name of the format. For more information, see "LICP0100 Format" on page 54-2.

Product identification format name
INPUT; CHAR(8)
The name of the format containing the information to identify the product. The format name is:

LICP0100 Basic product information used as input to the API. For details, see the "LICP0100 Format" on page 54-2.

License information
INPUT; CHAR(*)
Information that is used to license the product or feature. The structure of this information is determined by the

name of the format. For more information, see "LICI0100 Format" on page 54-2.

License information format name
INPUT; CHAR(8)
The name of the format containing the license information. The format name is:

LICI0100 Basic license information used as input to the API. For details, see "LICI0100 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.

LICP0100 Format

The following information uniquely describes the product or feature for which the license information is to be added. For detailed descriptions of the fields, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(7)	Product ID
7	7	CHAR(6)	Release
13	D	CHAR(4)	Feature

LICI0100 Format

The following specifies the format for the license information that is being added to the product or feature. For detailed descriptions of the fields, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(2)	Usage type
2	2	CHAR(2)	Compliance
4	4	BINARY(4)	Default usage limit
8	8	CHAR(1)	License term
9	9	CHAR(1)	Allow a license to be released

Field Descriptions

Allow a license to be released. Whether a use of the license that was previously requested can be released using the Work with License Information (WRKLICINF) command. When a use of the license is released, the usage count is decremented. This value can only be specified for a usage type of registered. The valid values are:

0 A use of the license cannot be released with the WRKLICINF command.

1 A use of the license can be released with the WRKLICINF command.

Compliance. The action taken when the usage limit is exceeded. The valid values are:

01 The usage limit cannot be exceeded. The user who attempts to use the product or feature after the usage limit has been reached is prevented from accessing the product or feature until the appropriate action is taken to increase the usage limit. A message indicating an attempt was made to exceed the usage limit is sent to the QSYSOPR message queue and to the message queues specified on the Change License Information (CHGLICINF) command.

02 The user who attempts to use the product or feature after the usage limit has been reached is allowed access. A warning message indicating that the usage limit has been exceeded is sent to the QSYSOPR message queue and to the message queues specified on the CHGLICINF command.

Default usage limit. The usage limit that will be in effect when the product or feature is initially installed. The valid values are:

0–999999 The number of users allowed to access the product or feature.

-1 Any number of users are allowed to access the product or feature.

Feature. The feature of the product to which the license information is being added. Valid values for the feature are 5001 through 9999.

License term. The length of time the authorized usage limit for a product lasts. The valid values are:

1 The authorized usage limit is valid for the entire version of the product or feature. Each time a new version is installed, the authorized usage limit must be set by using the Work with License Information (WRKLICINF) command or the Change License Information (CHGLICINF) command.

2 The authorized usage limit is valid for the entire release of the product or feature. Each time a new release is installed, the authorized usage limit must be set by using the WRKLICINF command or the CHGLICINF command.

3 The authorized usage limit is valid only for a modification of the product. Each time a new modification is installed, the authorized usage limit must be set by using the WRKLICINF command or the CHGLICINF command.

Product ID. The product ID of the product or feature to which the license information is being added.

Release. The version, release, and modification level of the product or feature to which the license information is to be added. The release level must be in the format V*x*R*y*M*z*. Valid values for *x* and *y* are 0 through 9, and valid values for *z* are 0 through 9 and A through Z.

Usage type. The type of license usage. The valid values are:

01 The usage limit is concurrent. It is for the number of unique jobs accessing the product or feature at one time.

02 The usage limit is registered. It is for the number of unique users registered by the product or feature.

Error Messages

CPF0C4B E Product availability object &2/&1 recovery required.
CPF0C4C E Unable to allocate product availability object &1 in library &2.
CPF0C4D E Unable to allocate product availability object &1 in library &2.
CPF0C54 E Data in product record not correct.
CPF0CB2 E Product ID &1 not valid. (Cannot use external command for internal product.)
CPF24B4 E Severe error while addressing parameter list.
CPF358A E Release not valid.
CPF3C21 E Format name &1 is not valid.
CPF3CF1 E Error code parameter not valid.
CPF8191 E Product definition &4 in &9 damaged.
CPF8193 E Product load object &4 in &9 damaged.
CPF9801 E Object &2 in library &3 not found.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9810 E Library &1 not found.
CPF9838 E User profile storage limit exceeded.
CPF9872 E Program &1 in library &2 ended. Reason code &3.
CPF9E04 E Product definition for product &1 &2 feature &3 cannot be found.
CPF9E05 E Feature &3 not valid.
CPF9E06 E Usage type not valid.
CPF9E07 E Compliance not valid.
CPF9E08 E Default usage limit not valid.
CPF9E09 E License term not valid.
CPF9E02 E Cannot add license information.
CPF9E03 E Cannot add license information.
CPF9E15 E Error in license management function.
CPF9E0A E Cannot specify Allow license release if license type *CONCURRENT.
CPF9E0C E Allow license release not valid.
CPF9E1A E License information conflict found.

Create Product Definition (QSZCRTPD) API

Parameters

Required Parameter Group:

1	Qualified product definition name	Input	Char(20)
2	Product definition information	Input	Char(106)
3	Product option list	Input	Array of Char(41)
4	Number of product options	Input	Binary(4)
5	Language load list	Input	Array of Char(20)
6	Number of language loads	Input	Binary(4)
7	Text description	Input	Char(50)
8	Public authority	Input	Char(10)
9	Error code	I/O	Char(*)

The Create Product Definition (QSZCRTPD) API creates a product definition (*PRDDFN) object. Each release of a packaged software product requires one product definition.

Authorities and Locks

Library Authority *ADD and *READ
Library Lock *SHRUPD
Product Availability Lock
*SHRRD. The product availability object resides in the QUSRSYS library.

Required Parameter Group

Qualified product definition name
INPUT; CHAR(20)
The first 10 characters contain the product definition name. The second 10 characters contain the name of the library into which the product definition is to be created. The following special value is supported for the library name:

***CURLIB** The job's current library

Product definition information
INPUT; CHAR(106)
A structure containing information about the product. For more information, see "Format of Product Definition Information" on page 54-4.

Product option list
INPUT; ARRAY of CHAR(41)
An array containing information for each of the options defined for the product. Each element of the array contains information for one option. There must be one element of the array for each option. The first element of the array must be for the base (0000) option. The required data for the product option list is described in "Format of Product Option List" on page 54-4.

Number of product options
INPUT; BINARY(4)
The number of options defined for the product. This

number is the same as the number of elements in the product option list parameter. Up to 100 product options can be specified. If the number of elements in the product option list is less than the value specified, the results are unpredictable.

Language load list
INPUT; ARRAY of CHAR(20)
Specifies which languages are defined for the product options. The required data for the language load list is described in "Format of Language Load List."

Number of language loads
INPUT; BINARY(4)
The number of elements in the language load list parameter. If the number of elements in the language load list parameter is less than the value specified, the results are unpredictable.

Text description
INPUT; CHAR(50)
Text that briefly describes the product definition object.

Public authority
INPUT; CHAR(10)
The authority you give to users who do not have specific authority to the product definition object and whose group profile has no specific authority to the object. Valid values are:

***ALL**
Allows the user to perform all operations on the object except those limited to the owner or controlled by the authorization list management authority.

***CHANGE**
Allows the user to perform all operations on the object except those limited to the owner or controlled by the object existence authority and object management authority.

***EXCLUDE**
Prevents the user from accessing the object.

***LIBCRTAUT**
The public authority for the object is taken from the value of the create authority (CRTAUT) parameter of the target library (the library that is to contain the object). This value is determined when the object is created. If the CRTAUT value for the library changes after the object is created, the new value does not affect any existing objects.

***USE**
Provides object operational authority and read authority.

Error code
I/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 Product Definition Information

The following table describes the order and format of the product definition information parameter. For detailed descriptions of fields in the table, see the "Field Descriptions" on page 54-5.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(7)	Product ID
7	7	CHAR(6)	Release level
13	D	CHAR(10)	Message file
23	17	CHAR(10)	First copyright
33	21	CHAR(10)	Current copyright
43	2B	CHAR(6)	Release date
49	31	CHAR(4)	Allow multiple releases
53	35	CHAR(10)	Registration ID type
63	3F	CHAR(14)	Registration ID value
77	4D	CHAR(29)	Reserved

Format of Product Option List

Up to 100 product options can be specified. The first option specified must be 0000 (the base option). The product option list parameter is described in the table below. The offsets shown in the table are for the first element in this array. For detailed descriptions of fields in the table, see the "Field Descriptions" on page 54-5.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(4)	Product option number
4	4	CHAR(7)	Message ID
11	B	CHAR(10)	Allow dynamic naming
21	15	CHAR(4)	Code load ID
25	19	CHAR(16)	Reserved

Format of Language Load List

Up to 30 language loads can be specified for the base (0000) option. The language load list parameter is described in the table below. The offsets shown in the table are for the first element in this array. For detailed descriptions of fields in the table, see the "Field Descriptions" on page 54-5.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(8)	Language load ID
8	8	CHAR(4)	Product option number
12	C	CHAR(8)	Reserved

Field Descriptions

Allow dynamic naming. Allow libraries and root folders to be named during installation of the product option. Valid values are:

**NODYNNAM* Do not allow naming of libraries and folders during installation.
**ALWDYNNAM* Allow naming of libraries and root folders during installation.

Allow multiple releases. Allow different releases of this product to be installed on the same system. Valid values are:

**NO* Do not allow multiple releases.
**YES* Allow multiple releases.

Code load ID. The identifier of the code load for the option. Valid values are 5001 through 9999. For all product options that are part of the same feature, specify the same code load ID. If you are not adding license information to your product, you should use 5001 for the code load ID for each option. The code load ID you specify for this option must be the same as the load ID specified when you create the code product load for this option. See "Create Product Load (QSZCRTPL) API" on page 54-6 for information about creating product loads.

Current copyright. The year of the most recent copyright for this product. The year must be specified as a 4-digit number, such as 1990. The field must be padded with blanks. If this field and the first copyright field both have values other than *NONE, then the first copyright field must be less than the current copyright field. The following special values are valid:

**CURRENT* The current year is used.
**NONE* The product does not have a current copyright.

First copyright. The first year that the product was copyrighted. The year must be specified as a 4-digit number, such as 1990. The field must be padded with blanks. The following special values are valid:

**CURRENT* The current year is used.
**NONE* The product does not have a first copyright.

Language load ID. The national language versions (NLVs) that are valid for a given product option. Individual NLVs may be specified for the base option, with one element of the language load list parameter for each NLV for the base option. For options other than the base option, only *BASEOPT and *NONE are valid. For example, to create a product definition with NLVs 2924 and 2931 defined for both the base option and option 1, the language load list would have three elements: one with 2924 0000, one with 2931 0000, and one with *BASEOPT 0001. Valid special values are:

**NONE* Defines the option to have no NLVs.
**IBMLNG* Only valid for the base option, 0000. Specifies that the product definition will be created with the list of all the NLVs for the base option being the same as that of the currently installed operating system.
**BASEOPT* Only valid for an option other than 0000. Defines this option to have the same NLVs as the base option of this product.

Message file. The name of the message file containing the messages that describe the product and its options. The message file for the base option is considered the message file for the product.

Message ID. The identifier of the message that describes the product option.

Product ID. The 7-character identifier of the product for which a product definition is being created. The product ID must be in the format *nlxxxxx*, where *n* is any numeric character 0 through 9, *l* is any uppercase letter A through Z, and *x* is any numeric character 0 through 9 or uppercase letter A through Z.

Product option number. The identifier of the product option.

When used in the product option list parameter, valid values are 0000 through 0099, with each number specified at most once. Specify 0000 for the base product option. The value 0000 must be the first option specified.

When used in the language load list parameter, this is the identifier for the product option for which NLVs are being defined. This must be one of the options specified on the product option list parameter.

Registration ID type. Specifies what the registration ID value field represents. Valid values are:

**PHONE* Telephone number will be entered in the registration ID value field.
**CUSTOMER* Country code and IBM customer number will be entered in the registration ID value field.

Registration ID value. The identifier of the organization to which the product belongs. This number should be unique from other vendors on the systems on which this product will be installed; it is recommended to specify either a telephone number (including the country code and city code) or your IBM customer number appended to your country code. Valid characters for the registration ID value are A through Z and 0 through 9, padded with blanks on the right.

Release date. Release date of the product. The format is yymmdd, where yy is the year, mm is the month, and dd is the day. The following special value is valid for the release date:

**NONE* The product does not have a release date.

Release level. The version, release, and modification level of the product being created in the format V*x*R*y*M*z*. Valid values for *x* and *y* are 0 through 9, and valid values for *z* are 0 through 9 or A through Z. For example, V2R1M0 is Version 2, Release 1, Modification 0.

Reserved. This field must contain blank characters; otherwise, an error occurs.

Error Messages

CPF0CAA E First product option not *BASE.
CPF0CAB E Language load identifier not valid.
CPF0CAC E Language load identifier not valid.
CPF0CAD E Duplicate language load identifier specified.
CPF0CAE E Language load not valid.
CPF0CAF E Duplicate option &3 specified.
CPF0CA3 E Code load &4 supported.
CPF0CA6 E Product definition &3 not created in library &4.
CPF0CBA E Number of options parameter not valid.
CPF0CBB E Release date &3 not valid.
CPF0CBC E Message identifier &4 not allowed.
CPF0CBD E Code load ID &9 not valid.
CPF0CB0 E Copyright dates not valid.
CPF0CB1 E Registration identifier not valid.
CPF0CB2 E Product identifier &1 not valid.
CPF0CB3 E Value for reserved field not valid.
CPF0CB5 E Copyright field &3 not valid.
CPF0CB6 E Allow multiple releases field not valid.
CPF0CB7 E Allow dynamic naming field not valid.
CPF0CB8 E Language load field not valid.
CPF0CB9 E Number of language loads parameter not valid.
CPF0C16 E Object &1 type &3 already exists in library &2.
CPF0C17 E *&3 object already exists for product &4 release &5.
CPF0C18 E Registration identifier &7 not valid for product &4 release &5.
CPF0C19 E Damage occurred on object &1 in library &2.
CPF0C4A E Product record not found.
CPF0C4D E Error occurred while processing object &1 in library &2.
CPF0C8A E Product option &1 not valid.
CPF0C84 E Load identifier &4 not valid.
CPF0C86 E Registration identifier not valid.
CPF0C9B E Authority &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C29 E Object name &1 is not valid.
CPF35E3 E Interface error detected.
CPF358A E Release not valid.
CPF9810 E Library &1 not found.
CPF9818 E Object &2 in library &3 not created.
CPF9819 E Object &2 in library &3 not created.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Create Product Load (QSZCRTPL) API

Parameters

Required Parameter Group:

1	Product load name	Input	Char(10)
2	Product load information	Input	Char(83)
3	Secondary language library name	Input	Char(10)
4	Principal library information	Input	Char(30)
5	Additional library list	Input	Array of Char(30)
6	Number of additional libraries	Input	Binary(4)
7	Preoperation exit programs	Input	Array of Char(20)
8	Number of preoperation exit programs	Input	Binary(4)
9	Folder list	Input	Array of Char(126)
10	Number of folders	Input	Binary(4)
11	Text description	Input	Char(50)
12	Public authority	Input	Char(10)
13	Error code	I/O	Char(*)

The Create Product Load (QSZCRTPL) API creates a product load (*PRDLOD) object. Each release of a software product requires one or more product load objects.

Authorities and Locks

Library Authority *ADD and *READ
Library Lock *SHRUPD
Product Availability Lock
*SHRRD. The product availability object resides in the QUSRSYS library.

Required Parameter Group

Product load name
INPUT; CHAR(10)
The name of the product load object to be created. The product load is created into the principal development library. The following special value is valid:

***LNG** The name of the load object is the same as the previously created language load object for this product; version, release, and modification level; and option. This field is only valid if a language product load is being created.

Product load information
INPUT; CHAR(83)
A structure containing information about the product load. For more information, see the "Format of Product Load Information" on page 54-8.

Secondary language library name
INPUT; CHAR(10)
The name of the secondary language library for the lan-

guage product load being created. This is the library into which this product load is installed if the language identifier for this product load does not match the system primary language identifier and if no override name is specified on the Restore Licensed Program (RSTLICPGM) command. This field is only valid if a language product load is being created.

Principal library information
INPUT; CHAR(30)
The first 10 characters specify the principal development library. The second 10 characters specify the principal primary library. The last 10 characters specify the postoperation exit program for both principal libraries. For more information, see "Format of Principal Library Information" on page 54-8.

Additional library list
INPUT; ARRAY of CHAR(30)
The additional libraries for the product load. Additional libraries do not need to exist before the product load object is created. For each element, the first 10 characters specify the development library, the second 10 characters specify the primary library, and the last 10 characters specify the postoperation exit program for both libraries. For more information, see "Format of Additional Library List" on page 54-8.

Number of additional libraries
INPUT; BINARY(4)
The number of elements in the additional library list. If the number of elements in the additional library list is less than the value specified, the results are unpredictable.

Preoperation exit programs
INPUT; ARRAY of CHAR(20)
The preoperation exit programs for this load. The first 10 characters specify the exit program name. The second 10 characters specify the development library name. For more information, see "Format of Preoperation Exit Programs" on page 54-8.

Number of preoperation exit programs
INPUT; BINARY(4)
The number of elements in the preoperation exit programs array. If the number of elements in the preoperation exit programs parameter is less than the value specified, the results are unpredictable.

Folder list
INPUT; ARRAY of CHAR(126)
The folders for this product load. When creating a code load, the first folder specified must be a root folder. When creating a language load, the first folder specified must be a subfolder of a root folder. The folders do not need to exist before the product load object is created.

Each product option has at most one root folder. A folder cannot belong to more than one product option. The root folder must be part of the code load. Folders must be specified so that a parent folder precedes its subfolder on the list. A maximum of 100 folders can be specified for a load. The documents in the development folders are saved when the product load is saved with the Save Licensed Program (SAVLICPGM) command. For more information, refer to the *SystemView* System Manager/400 User's Guide*.

For each element of the folder list, the first 63 characters specify the development folder and the next 63 characters specify the primary folder. For more information, see "Format of Folder List" on page 54-8.

Number of folders
INPUT; BINARY(4)
The number of elements in the folder list array. If the number of elements in the folder list is less than the value specified, the results are unpredictable.

Text description
INPUT; CHAR(50)
Text that briefly describes the product load object.

Public authority
INPUT; CHAR(10)
The authority you give to users who do not have specific authority to the product load object and whose group profile has no specific authority to the object. Valid values are:

***ALL**
Allows the user to perform all operations on the object except those limited to the owner or controlled by the authorization list management authority.

***CHANGE**
Allows the user to perform all operations on the object except those limited to the owner or controlled by the object existence authority and object management authority.

***EXCLUDE**
Prevents the user from accessing the object.

***LIBCRTAUT**
The public authority for the object is taken from the value of the create authority (CRTAUT) parameter of the target library (the library that is to contain the object). This value is determined when the object is created. If the CRTAUT value for the library changes after the object is created, the new value does not affect any existing objects.

***USE**
Provides object operational authority and read authority.

Error code
I/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 Product Load Information

The product load information parameter is described in the following table. For a detailed description of the fields in the table, see "Field Descriptions" on page 54-8.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(7)	Product ID
7	7	CHAR(6)	Release level
13	D	CHAR(4)	Product option
17	11	CHAR(10)	Product load type
27	1B	CHAR(8)	Load ID
35	23	CHAR(10)	Registration ID type
45	2D	CHAR(14)	Registration ID value
59	3B	CHAR(24)	Reserved

Format of Principal Library Information

The principal library information parameter is described in the following table. For a detailed description of the fields in the table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Principal development library name
10	A	CHAR(10)	Principal primary library name
20	14	CHAR(10)	Postoperation exit program name

Format of Additional Library List

The following table describes the additional library list parameter. The offsets shown in the table are for the first element in this array. For a detailed description of the fields in the table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Additional development library name
10	A	CHAR(10)	Additional primary library name
20	14	CHAR(10)	Postoperation exit program name

Format of Preoperation Exit Programs

The following table describes the preoperation exit programs parameter. The offsets shown in the table are for the first element in this array. For a detailed description of the fields in the table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Preoperation exit program name
10	A	CHAR(10)	Development library name

Format of Folder List

The following table describes the folder list parameter. The offsets shown in the table are for the first element in this array. For a detailed description of the fields in the table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(63)	Development folder
63	3F	CHAR(63)	Primary folder

Field Descriptions

Additional development library name. The name of the additional development library.

Additional primary library name. The additional primary library name. Valid special values are:

**DVLLIB* The development library name is used as the primary library name.

**CODE* The additional primary library in the code load corresponding to the immediately preceding development library is used. This value is valid only when the product load type is specified as *LNG in the product load information parameter.

Development folder. The name of a development folder for this load. The folder does not need to exist before the product load object is created.

Development library name. The development library with which the preoperation exit program is associated. If the preoperation exit program is to be associated with the principal library, then this must be the same as the value specified for the principal development library in the principal library information parameter. If the preoperation exit program is to be associated with an additional library, then this must be the same as one of the values specified for the additional development library in the additional library list parameter. Valid special values are:

*PRDDFN Specify this value if you want the preoperation exit program to be associated with the principal development library and if you specified *PRDDFN for the principal development library field of the principal library information parameter. This is only valid when *PRDDFN is specified for the principal development library in the principal library information parameter.

*CODE Specify this value if you want the preoperation exit program to be associated with the principal development library and if you specified *CODE as the principal development library field of the principal library information parameter. This is only valid when *CODE is specified for the principal development library in the principal library information parameter.

Load ID. The load ID of the product load to be created. For language loads, this must be a valid national language version (NLV). The following special value is valid:

*CODEDFT The default code load ID, 5001, is used. This value is valid only when the product load type field is *CODE.

Postoperation exit program name. The program that is called in the corresponding installed library after any of these operations are performed on the product load:

- Save, using the Save Licensed Program (SAVLICPGM) command
- Restore, using the Restore Licensed Program (RSTLICPGM) command
- Check, using the Check Product Option (CHKPRDOPT) command

The exit program does not need to exist before the product load object is created. The following special value is valid:

*NONE No exit program is called after the library is saved, restored, or checked.

Preoperation exit program name. The name of the preoperation exit program. A preoperation exit program is called before any of these operations are performed on the library:

- Save, using the Save Licensed Program (SAVLICPGM) command
- Restore, using the Restore Licensed Program (RSTLICPGM) command
- Delete, using the Delete Licensed Program (DLTLICPGM) command.

The exit program does not need to exist before the product load object is created.

Primary folder. The primary folder name associated with the development folder. The following special value is valid:

*DVLFLR The development folder name is the same as the primary folder name.

Principal development library name. The library into which the product load is created. Valid special values are:

*PRDDFN The name of the library in which the product definition exists is used for the development library name.

*CODE The name of the principal development library for the code load is used. This value is valid only when the product load type field is specified as *LNG.

Principal primary library name. The product load is installed into this library when no override name is specified on the Restore Licensed Program (RSTLICPGM) command. Valid special values are:

*DVLLIB The development library name is used as the primary library name.

*CODE The name of the principal development library for the code load is used. This value is valid only when the product load type field is specified as *LNG.

Product ID. The 7-character identifier of the product for which a product load is being created. The product ID must be in the format *nlxxxxx*, where *n* is any numeric character 0 through 9, *l* is any uppercase letter A through Z, and *x* is any numeric character 0 through 9 or uppercase letter A through Z.

Product load type. Whether the product load being created is a code load or a language load. Valid values are:

*CODE A code load is created.

*LNG A language load is created.

Product option. The product option for which a product load is being created. Use 0000 for the base option.

Registration ID type. Specifies what the registration ID value field represents. Valid values are:

*PRDDFN The registration ID is taken from the product definition for this product and release level. The product definition must exist for *PRDDFN to be valid.

*PHONE A telephone number will be entered in the registration ID value field.

*CUSTOMER The country code and IBM customer number will be entered in the registration ID value field.

Registration ID value. Identifier of the organization to which the product belongs. This number should be unique from other vendors on the systems on which this product will be installed. It is recommended that you specify a telephone number including the country and city code, or specify your country code followed by your IBM customer number. Valid characters for the registration ID value are A through Z and 0 through 9, padded with blanks on the right.

Release level. The version, release, and modification level of the product being created in the format V*x*R*y*M*z*. Valid values for *x* and *y* are 0 through 9, and valid values for *z* are

0 through 9 or A through Z. For example, V2R1M0 is Version 2, Release 1, Modification 0.

Reserved. This field must contain blank characters; otherwise, an error occurs.

Error Messages

CPF0CA1 E No language load defined.
CPF0CA2 E Code load ID &9 not valid.
CPF0CA3 E Code load &4 supported.
CPF0CB1 E Registration identifier not valid.
CPF0CB2 E Product identifier &1 not valid.
CPF0CB3 E Value for reserved field not valid.
CPF0C16 E Object &1 type &3 already exists in library &2.
CPF0C17 E *&3 object already exists for product &4 release &5.
CPF0C18 E Registration identifier &7 not valid for product &4 release &5.
CPF0C19 E Damage occurred on object &1 in library &2.
CPF0C4A E Product record not found.
CPF0C4B E Product availability object &2/&1 recovery required.
CPF0C4C E Cannot allocate object &1 in library &2.
CPF0C4D E Error occurred while processing object &1 in library &2.
CPF0C54 E Data in product record not correct.
CPF0C8A E Product option &1 not valid.
CPF0C8B E Product load type &1 not valid.
CPF0C8C E Number of additional libraries not valid.
CPF0C8D E Preoperation exit program information not valid.
CPF0C8E E Preoperation exit program library not valid.
CPF0C8F E Number of folders not valid.
CPF0C81 E Product load &6 in library &5 not created.
CPF0C82 E Error occurred while creating product load &6 in library &5.
CPF0C83 E Previous level folder not specified.
CPF0C84 E Load identifier &4 not valid.
CPF0C85 E Duplicate library &5 specified.
CPF0C87 E Library &1 not allowed.
CPF0C9B E Authority &1 not valid.
CPF0C9C E Secondary language library name required.
CPF0C91 E Code load does not exist.
CPF0C92 E Folder name not correct.
CPF0C93 E More than one root folder specified.
CPF0C94 E Object name *LNG not valid for code load.
CPF0C95 E *CODE not valid for library.
CPF0C96 E Secondary language library not valid.
CPF0C97 E Duplicate folder &5 in folder list.
CPF0C98 E Additional development library &5 not found in code load.
CPF0C99 E Product definition object not found.
CPF0613 E User profile does not have enough storage assigned.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C29 E Object name &1 is not valid.
CPF358A E Release not valid.
CPF9810 E Library &1 not found.
CPF9818 E Object &2 in library &3 not created.
CPF9819 E Object &2 in library &3 not created.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Create Program Temporary Fix (QPZCRTFX) API

Parameters

Required Parameter Group:

1	PTF information	Input	Char(50)
2	Development library name	Input	Char(10)
3	Objects	Input	Array (*) of Char (20)
4	Number of objects	Input	Binary(4)
5	Documents	Input	Array (*) of Char(73)
6	Number of documents	Input	Binary(4)
7	Prerequisite PTFs	Input	Array(*) of Char(24)
8	Number of prerequisite PTFs	Input	Binary(4)
9	Exit programs	Input	Array(*) of Char(84)
10	Number of exit programs	Input	Binary(4)
11	Problem IDs	Input	Array(*) of Char(10)
12	Number of problem IDs	Input	Binary(4)
13	Cover letters	Input	Array(*) of Char(44)
14	Number of cover letters	Input	Binary(4)
15	Error code	I/O	Char(*)

This API should only be used by organizations to create program temporary fixes (PTFs) for products that they develop.

The Create Program Temporary Fix (QPZCRTFX) API creates a PTF save file and optionally creates cover letters in the general purpose library (QGPL). The save file contains a PTF control object and any number of fix objects. The save file name is the PTF identifier preceded by the letter Q. If a file with the same name already exists in QGPL, a unique name is generated by the system. This name is a timestamp preceded by the letter Q. After creating the PTF, you can use the Display PTF (DSPPTF) command to view the PTF attributes.

PTFs can only be created for products that are installed.

PTFs must be created by a profile that is known to exist on all systems so that the PTF can be loaded on any system that has the product installed.

Authorities and Locks

Cover Letter Source File Authority
*USE

| **Cover Letter Source File Library Authority**
| *USE
| **Object Authority** *CHANGE
| **Object Library Authority**
| *USE
| **QAPZCOVER Authority**
| *ALL
| **QGPL Library Authority**
| *CHANGE

| This API does not adopt authority.

| Required Parameter Group

| **PTF information**
| INPUT; CHAR(50)
| Attributes of the PTF to be created. See "PTF Information Format" on page 54-12 for more information about this field.

| **Development library name**
| INPUT; CHAR(10)
| The library in which the fix is located. This can be any library.

| **Objects**
| INPUT; ARRAY(*) of CHAR(20)
| The name and type of each object to be included in the PTF. The first 10 characters contain the name, and the second 10 characters contain the external type of the object.

| **Object name**
| The name of the object.
| **Object type** The external type of the object. This must be preceded by an asterisk (*). For more information, refer to the *SystemView* System Manager/400 User's Guide*.

| **Number of objects**
| INPUT; BINARY(4)
| The number of objects listed in the objects parameter. This number must be in the range of 1 through 300.

| **Documents**
| INPUT; ARRAY(*) of CHAR(73)
| The name of the documents that are to be included in the PTF.

| The create PTF function copies the document from a subfolder using the name specified, followed by /QP. For example, if a PTF is being created for a product folder called PRODUCT, the fix objects must be developed in a subfolder named PRODUCT/QP. The document is installed into the product folder PRODUCT during the apply PTF operation. The QP subfolder allows you to develop a PTF without changing the product.

| **Document name**
| The name of the document including the path name.

| **Number of documents**
| INPUT; BINARY(4)
| The number of documents listed in the documents parameter. This number must be in the range of 1 through 300.

| **Prerequisite PTFs**
| INPUT; ARRAY(*) of CHAR(24)
| The list of prerequisite PTFs. A prerequisite relationship exists when one PTF requires that another PTF also be applied. Prerequisite PTFs must exist within the same product. A prerequisite PTF must already exist on the system or the create operation will fail. The first 7 characters are the PTF ID of the prerequisite PTF. The rest of the field is reserved.

| **Number of prerequisite PTFs**
| INPUT; BINARY(4)
| The number of PTFs included in the prerequisite PTFs parameter. This number must be in the range of 1 through 300.

| **Exit programs**
| INPUT; ARRAY(*) of CHAR(84)
| The PTF exit programs called when a PTF is temporarily applied, permanently applied, temporarily removed, or permanently removed. Exit programs eliminate the need for you to manually carry out special instructions to install the PTF. The run option field of this parameter determines when the exit program is called.

| Shipping the same exit program in two PTFs causes one PTF to supersede the other.

| For more information on this structure, see "Exit Programs Format" on page 54-12 and "Program Temporary Fix Exit Program" on page 55-2.

| **Number of exit programs**
| INPUT; BINARY(4)
| The number of exit programs listed in the exit programs parameter. This number must be in the range of 1 through 50.

| **Problem IDs**
| INPUT; ARRAY(*) of CHAR(10)
| A list of the problem IDs for problems that this PTF fixes. By listing the problem IDs, the symptom strings associated with those problems will be included in the PTFs.

| **Number of problem IDs**
| INPUT; BINARY(4)
| The number of problem IDs listed in the problem IDs parameter. This number must be in the range of 1 through 300.

| **Cover letters**
| INPUT; ARRAY(*) of CHAR(44)
| A cover letter can be created for each of the national language versions (NLV) that IBM supports. A member that contains source for each PTF cover letter must be supplied as input to the API. The cover letter file can be a source file with a maximum record length of 92 or a physical file with record length of 80. The cover letter

must be in the file before this API is called. Only one cover letter per NLV is allowed.

For more information on this structure see "Cover Letter Format."

Number of cover letters
INPUT; BINARY(4)
The number of cover letters listed in the cover letter parameter. This number must be in the range of 1 through 50.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

PTF Information Format

For detailed descriptions of each field, see the "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(7)	PTF ID
7	7	CHAR(7)	Product ID
14	E	CHAR(6)	Release level
20	14	CHAR(4)	Product option
24	18	CHAR(10)	Primary object library name
34	22	CHAR(4)	Load ID
38	26	CHAR(12)	Reserved

Exit Programs Format

Each of the 50 elements in the array for the exit programs parameter has the following format. The information must be presented in the order listed below. The exact offsets for each entry are not given. For detailed descriptions of each field, see the "Field Descriptions."

Offset			
Dec	Hex	Type	Field
		CHAR(10)	Exit program name
		CHAR(10)	Exit program library name
		CHAR(7)	Run option
		CHAR(7)	Exit program type
		CHAR(50)	User data

Cover Letter Format

Each of the 50 elements in the array for the cover letter parameter has the following format. The information must be presented in the order listed below. The exact offsets for each entry are not given. For detailed descriptions of each field, see the "Field Descriptions."

Offset			
Dec	Hex	Type	Field
		CHAR(10)	Cover letter file name
		CHAR(10)	Cover letter library name
		CHAR(10)	Cover letter member name
		CHAR(4)	NLV
		CHAR(10)	Reserved

Field Descriptions

Cover letter file name. The name of the file where the cover letter can be located.

Cover letter library name. The name of the library where the cover letter file can be located.

Cover letter member name. The member name that contains the cover letter.

Exit program library name. The library where the exit program can be found if it is not included in the PTF.

Exit program name. The name of the exit program.

Exit program type. Whether the exit program is to be included in this PTF. The possible values are:

**PTF* The exit program is to be included in the PTF. The exit program must exist in the library specified in the exit program library field.
**OBJLST* The exit program is part of the product and should not be included in the PTF. The exit program must exist in the object list for the product, option, and load of the PTF being created or in the principal library for the base option (*BASE) of the product.

Load ID. The load ID of the product load for the PTF. This will be a language load if the PTF is for textual data, or it will be the code load.

NLV. The NLV of the cover letter This must be a valid AS/400 NLV.

Primary object library name. The library in which the objects are to be placed when the PTF is installed. If necessary, the PTF apply operation overrides the PTF library specified when the PTF was created. Two cases where this might occur are:

- The product load has been installed into a secondary language library.
- Dynamic library renaming was used when the product was installed.

Product ID. The product for which the PTF is being created. This product must be installed on the system.

Product option. The option of the product for which the PTF is being created. All objects in the PTF must be for the same option.

PTF ID. The ID the PTF is to be known by. The identifier must be 7 characters. The first character must be numeric. The second and third characters must be alphabetic. The same identifier can be used only once for each product and release level.

Release level. The version, release, and modification level of the product in the format V*x*R*y*M*z*. Valid values for *x* and *y* are 0 through 9, and valid values for *z* are 0 through 9 or A through Z.

Reserved. An error will be signaled if this field does not contain blanks.

Run option. When the exit program is to be run. The possible values are:

**BOTH* The exit program will be run when the PTF is applied and removed.
**APPLY* The exit program will be run when the PTF is applied.
**REMOVE* The exit program will be run when the PTF is removed.

User data. Any data you want to pass to the exit program.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C29 E Object name &1 is not valid.
CPF3C31 E Object type &1 is not valid.
CPF35BC E Object type &1 not supported.
CPF35CC E Library required for building PTFs already exists.
CPF35DA E Folder &1 not valid.
CPF35DB E Duplicate documents specified.
CPF35DC E Primary library not found.
CPF35DD E Problem &1 does not exist.
CPF35D3 E Cover letter not copied.
CPF35D4 E Cover letter file record length too long.
CPF35D5 E Cover letter NLV not valid.
CPF35D6 E Duplicate exit programs specified.
CPF35D7 E Required PTF &1-&2 &3 not valid.
CPF35D8 E Exit program &1 not valid.
CPF35D9 E Duplicate objects specified.
CPF357A E Parameter value not valid.
CPF357B E Product not found.
CPF357D E Document or folder name not correct.
CPF3570 E No PTF IDs available in range.
CPF3571 E PTF ID &1 not within valid range.
CPF3572 E PTF &2-&1 &3 already exists.
CPF3573 E Resources required for product &1 are not available.
CPF3574 E PTF ID not valid.
CPF358A E Release not valid.
CPF358B E PTF not created.
CPF358C E Create PTF not allowed for product &1.
CPF358D E Run option not valid.
CPF358E E Exit program type not valid.
CPF3901 E PTF &1-&2 &3 not created.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Delete Product Definition (QSZDLTPD) API

Parameters

Required Parameter Group:

1	Qualified product definition name	Input	Char(20)
2	Error code	I/O	Char(*)

The Delete Product Definition (QSZDLTPD) API deletes a single product definition object. Note that the product definition for a product and release cannot be deleted using this API when the product is supported by the Work with Supported Products (WRKSPTPRD) command. If the product is supported, the support must be removed before the product definition object can be deleted using this API. The WRKSPTPRD command is part of the SystemView System Manager/400 licensed program.

Authorities and Locks

Library Authority *READ
Library Lock *SHRUPD
Object Lock *EXCL
Product Definition Authority *OBJEXIST

Required Parameter Group

Qualified product definition name
INPUT; CHAR(20)
The first 10 characters contain the product definition object name, and the second 10 characters contain the name of the library where the product definition object is located.

Error code
I/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

CPF0C4B E Product availability object &2/&1 recovery required.
CPF0C52 E Product definition &3 in library &4 not deleted.
CPF2105 E Object &1 in &2 type *&3 not found.
CPF2110 E Library &1 not found.
CPF2113 E Cannot allocate library &1.
CPF2114 E Cannot allocate object &1 in &2 type *&3.
CPF2125 E No objects deleted.
CPF2176 E Library &1 damaged.
CPF2182 E Not authorized to library &1.
CPF2189 E Not authorized to object &1 in &2 type *&3.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C29 E Object name &1 is not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Delete Product Load (QSZDLTPL) API

Parameters

Required Parameter Group:

1	Qualified product load name	Input	Char(20)
2	Error code	I/O	Char(*)

The Delete Product Load (QSZDLTPL) API deletes a single product load object.

Authorities and Locks

Library Authority *READ
Library Lock *SHRUPD
Object Lock *EXCL
Product Load Authority *OBJEXIST

Required Parameter Group

Qualified product load name
INPUT; CHAR(20)
The first 10 characters contain the product load object name, and the second 10 characters contain the name of the library where the product load object is located.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF2105 E Object &1 in &2 type *&3 not found.
CPF2110 E Library &1 not found.
CPF2113 E Cannot allocate library &1.
CPF2114 E Cannot allocate object &1 in &2 type *&3.
CPF2125 E No objects deleted.
CPF2176 E Library &1 damaged.
CPF2182 E Not authorized to library &1.
CPF2189 E Not authorized to object &1 in &2 type *&3.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C29 E Object name &1 is not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Generate Program Temporary Fix Name (QPZGENNM) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	PTF information	Input	Char(50)
4	Format name	Input	Char(8)
5	Error code	I/O	Char(*)

PTF save files and cover letters are usually named Q plus the PTF ID. In cases where a PTF for another product exists on the system and has the same ID, another name must be selected for the PTF save file and the cover letter member name. The first 8 characters of the cover letter member name must match the PTF save file name. The Generate PTF Name (QPZGENNM) API generates a save file or cover letter member name for a PTF. Checking is done to verify that the name has not been used. A unique cover letter name is generated for each national language version (NLV).

You can use the QPZGENNM API to:

- Generate a unique name for a PTF save file in the library returned in the library name field.
- Generate a unique name for a PTF cover letter member name in the library and file returned in the library name field and file name field.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The receiver variable to receive the generated name. 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.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. The length must be at least 8 bytes. If this value is larger than the actual receiver variable, unexpected results may occur.

PTF information
INPUT; CHAR(50)
The information about the PTF or cover letter needed to generate the name. For more information, see "Format of PTF Information" on page 54-15.

Format name
INPUT; CHAR(8)
The format of the returned information.

PTFG0100 Save file and library name.
PTFG0200 Cover letter member, file, and library name.

For more information, see "Format of Returned Information."

Error code
I/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 PTF Information

The following describes the PTF information parameter. For a detailed description of the fields in this table, see "Field Descriptions."

Offset		Type	Field
Dec	Hex		
0	0	CHAR(7)	PTF ID
7	7	CHAR(7)	Product ID
14	E	CHAR(6)	Release
20	14	CHAR(14)	Reserved
34	22	CHAR(4)	National language version
38	26	CHAR(12)	Reserved

Format of Returned Information

When you generate a name for a PTF or cover letter, the name is only unique in the library and file returned by the API. The formats below show the fields returned in the receiver variable parameter. For a detailed description of the fields in these tables, see "Field Descriptions."

PTFG0100 Format

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	File name
18	12	CHAR(10)	Library name
28	26	CHAR(*)	Reserved

PTFG0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(28)	Everything from the PTFG0100 format
28	1C	CHAR(10)	Member name
38	26	CHAR(*)	Reserved

Field Descriptions

Bytes available. The number of bytes of data available to be returned to the user.

Bytes returned. The number of bytes that were returned to the user. This is the lesser of the number of bytes available to be returned or the length of the receiver variable.

File name. If the format requested was PTFG0100, this is the save file name to be used to store the PTF. If the format requested was PTFG0200, this is the file where the cover letter should be stored.

Library name. The library name for the PTF or cover letter file.

Member name. The member name for the cover letter.

National language version. The national language version of the cover letter for which the member name is to be generated. This field is ignored when the format is PTFG0100.

Product ID. The PTF is for this product.

PTF ID. The identifier of the PTF.

Release. The version, release, and modification level of the PTF. This must be in the format V*x*R*y*M*z*. Valid values for *x* and *y* are 0 through 9, and valid values for *z* are 0 through 9 or A through Z.

Reserved. This field is ignored in the returned variable parameter. In the PTF information parameter, this field is reserved and must contain blanks.

Usage Notes

The file name is not reserved and could be in use when a succeeding function tries to use it.

A name cannot be generated for a PTF that is already in device *SERVICE. An error will be signaled if this is done. For a description of *SERVICE, see the "Log Program Temporary Fix Information (QPZLOGFX) API" on page 54-16.

A name cannot be generated for a cover letter member if a cover letter for the same NLV and PTF ID already exists.

Error Messages

CPF0CB3 E Value for reserved field not valid.
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.
CPF3C24 E Length of the receiver variable is not valid.
CPF35BE E Product &1 &3 not supported or installed.
CPF35D5 E Cover letter NLV not valid.
CPF35ED E PTF ID &1 not valid.
CPF35E9 E PTF &1-&2 &3 already in *SERVICE.
CPF358A E Release not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Log Program Temporary Fix Information (QPZLOGFX) API

Parameters

Required Parameter Group:

1	PTF Information	Input	Char(50)
2	Request type	Input	Char(10)
3	Qualified file name	Input	Char(20)
4	Member name	Input	Char(10)
5	Error code	I/O	Char(*)

The Log PTF Information (QPZLOGFX) API allows you to specify the device from which PTFs are loaded as *SERVICE. You can use the QPZLOGFX API to indicate that a PTF or cover letter should be put into device *SERVICE.

PTFs are put into *SERVICE when they are received on the system using the Copy PTF Save File (CPYPTFSAVF) command and the Send PTF Order (SNDPTFORD) command. These commands put information about the PTF into the PTF database files so that the PTF can be displayed with the Display PTF (DSPPTF) command and loaded from device *SERVICE using the Load PTF (LODPTF) command. PTFs received on the system by any other method are not in *SERVICE.

The PTF must exist in a save file with a valid name in the designated PTF library before this API is called. If the PTF is put in a save file with a name that is being used by another PTF, an error (CPF35BD) will occur; the save file must be deleted or renamed. Checking is done to ensure that the correct library is being used.

The cover letter must exist in the designated PTF cover letter file with a valid member name before this API is called. If the member name is being used by another PTF, an error (CPF35FE) will occur; you must delete the member or rename it.

The product the PTF is for must be installed or supported.

The save file or member name must be Q plus the PTF ID. If that name is being used by another PTF that is in *SERVICE, you must select another name.

Required Parameter Group

PTF information
INPUT; CHAR(50)
The information needed to put the PTF into device *SERVICE. If the information specified here does not match the information in the member or save file specified on the qualified file name parameter, an error will occur (CPF35E6). A check is done to verify that the PTF ID, product ID, and release specified here match the PTF ID, product ID, and release of the cover letter or PTF. For more information about this parameter see "PTF Information Format" on page 54-17.

Request type
INPUT; CHAR(10)
The type of log operation to be done. The possible values are:

***LOGPTF** The PTF is to be put in device *SERVICE. The PTF must exist in the save file specified on the qualified file name parameter because product information from the save file is checked against the product information passed into this API.

***LOGCVR** A cover letter is to be put in device *SERVICE. The cover letter must exist in the library, file, and member specified in the qualified file name and member name parameters. Information from the member is checked against the product information passed into the API.

Qualified file name
INPUT; CHAR(20)
The file where the cover letter or PTF is located. If the request type parameter is *LOGPTF, then this is the save file name and library name where the PTF is located. If the request type is *LOGCVR, then this is the file and library that contain the cover letter. The first 10 characters are the file name and the second 10 characters are the library name.

Member name
INPUT; CHAR(10)
The member that contains the cover letter. This parameter is ignored if the request type parameter is *LOGPTF.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

PTF Information Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(7)	PTF ID
7	7	CHAR(7)	Product ID
14	E	CHAR(6)	Release level
20	14	CHAR(30)	Reserved

Field Descriptions

Product ID. The product that the PTF or the cover letter is for.

PTF ID. The identifier of the PTF.

Release level. The version, release, and modification level of the PTF. The release must be in the format V*x*R*y*M*z*. Valid values for *x* and *y* are 0 through 9, and valid values for *z* are 0 through 9 or A through Z.

Reserved. This field must be blank.

Error Messages

CPF0CB3 E Value for reserved field not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C20 E Error found by program &1.
CPF35BD E File name &4 not valid for PTF &1-&2 &3.
CPF35BE E Product &1 &3 not supported or installed.
CPF35BF E File name or library not valid.
CPF35DE E Request type &1 not valid.
CPF35E5 E PTF save file &1 in library &2 does not exist.
CPF35E6 E File or member not processed.
CPF35FE E Member name &7 not valid for PTF &1-&2 &3.
CPF358A E Release not valid.
CPF3903 E Cover letter not found.
CPF3905 E Member not valid cover letter.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Log Software Error (QPDLOGER) API

Parameters

Required Parameter Group:

1	Suspected program name	Input	Char(10)
2	Detection ID	Input	Char(12)
3	Message reference key	Input	Char(4)
4	Point of failure	Input	Binary(4)
5	Print job log	Input	Char(1)
6	Data items	Input	Char(*)
7	Data item offset and length	Input	Array of Char(*)
8	Number of data items	Input	Binary(4)
9	Object name	Input	Array of Char(*)
10	Number of object names	Input	Binary(4)
11	Error code	I/O	Char(*)

Optional Parameter:

12	ILE module name	Input	Char(10)

The Log Software Error (QPDLOGER) API allows a program to report a software problem to the local AS/400 system and provide the data needed to resolve the problem. When this API is called, any error data provided is spooled to one or more spooled files, a symptom string is created, an entry is created in the problem log, and a message is sent to the QSYSOPR message queue indicating that a software error has been detected.

Error data can be provided on the API call by using the data item offset and length and object name parameters.

When this API is called, the following informational messages can be placed in the job log:

CPI93CA Suspected program &1 not found.
CPI93CB Point-of-failure value not valid.
CPI93CC Object &1 in library &2 not found.
CPI93CF Data length or data offset not valid.

Required Parameter Group

Suspected program name
INPUT; CHAR(10)
The name of the program in which the error is suspected. This can be the reporting program or a different program. Valid values are:

***SAME** The reporting program.
***PRV** The program that called the reporting program.
program name
The name of the suspected program.

The suspected program name is included in the symptom string (as F/*name*) created when this API is called.

Detection ID
INPUT; CHAR(12)
A message ID or other value defined by the reporting

program that further identifies the problem. This value is included in the symptom string (as MSG*detectionid*) created when this API is called.

Message reference key
INPUT; CHAR(4)
The message key associated with the message that is being reported (if a message is being reported). This parameter is used to verify that message CPF9999 (a function check) was not caused by a damage exception (CPF81xx). If message CPF9999 is caused by a damage exception, the problem will not be reported. This value is ignored if it does not contain a key for a CPF9999 message.

Point of failure
INPUT; BINARY(4)
A return code, statement number, or other value defined by the reporting program that assists in locating the problem. This value is included in the symptom string (as RC*nnnnnnnn*) created when this API is called.

Print job log
INPUT; CHAR(1)
Whether the job log and other job information is to be spooled to a spooled file. Valid values are:

Y Print the job log and job information.
N Do not print the job log and job information.

Data items
INPUT; CHAR(*)
The data to be spooled.

Data item offset and length
INPUT; ARRAY of CHAR(*)
An array of the offsets to and lengths of the data items to be spooled to a spooled file. The array can contain up to 32 elements. Each element has the following structure:

Data offset BINARY(4). The offset to the data item from the start of the data.
Data length BINARY(4). The length of this data item (must be greater than 0).

Number of data items
INPUT; BINARY(4)
The number of elements in the array of data item offsets and lengths. The number must be between 0 and 32, inclusive.

Object name
INPUT; ARRAY of CHAR(*)
An array of object names whose contents are to be spooled to a spooled file. The array can contain up to 32 elements. Each element has the following structure:

Object name
CHAR(30). The name of the object to be spooled.
Library CHAR(30). The library in which the object resides. Valid values for the library name are:

***CURLIB** The job's current library.
***LIBL** The library list.
library name The specific library that contains the object.

Object type CHAR(10). The object type. For a complete list of the available object types, see the *Programming Reference Summary*.

Number of object names
INPUT; BINARY(4)
The number of object names in the array of object names. The number must be between 0 and 32, inclusive.

Error code
I/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.

Optional Parameter

ILE module name
INPUT; CHAR(10)
The name of the integrated language environment (ILE) module in which the error is suspected. This value is included in the symptom string created when this API is called.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF93C0 E Software error logging not active.
CPF93C2 E &1 is not a valid number of data items.
CPF93C3 E &1 is not a valid number of object names.
CPF93C4 E Error already logged.
CPF93C5 E Software problem logging (QPDLOGER) API error occurred.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Package Product Option (QSZPKGPO) API

Parameters

Required Parameter Group:

1	Product option information	Input	Char(35)
2	Repackage	Input	Char(4)
3	Allow object change	Input	Char(5)
4	Error code	I/O	Char(*)

The Package Product Option (QSZPKGPO) API packages one or more product loads for a specified product option, enabling the loads to be saved using the Save Licensed Program (SAVLICPGM) command. Object lists are added to the product load (*PRDLOD) object when the product load is packaged. In order to package a product load, all develop-

ment libraries for the load must exist and all development folders for the load must exist. Also, all objects to be packaged must exist and all objects other than folders and documents must have the correct product ID, release level, product option, and load ID in the object description. The object description can be changed using the Change Object Description (QLICOBJD) API.

To package a product load that has folders, the user must be enrolled in the system distribution directory and must have *ALL authority to the folders.

Other than these restrictions, a user who has authority to this API can package any product load created in either of the following ways, regardless of whether the user has authority to the objects for the product load.

- The Create Product Load (QSZCRTPL) API
- SystemView System Manager/400 licensed program

Authorities and Locks

Authority to Folders
*ALL

Authority to Objects Packaged
None

Document Lock
*SHRRD. A lock of *SHRNUP is obtained on each document's attributes.

Table of Contents for Subordinate Folders Lock
*SHRNUP

Folder Lock *SHRUPD

Object Lock *EXCLRD. If *NO is specified for the allow object change parameter, each object packaged is locked *EXCLRD.

Product Availability Lock
*SHRRD

Product Definition Lock
*EXCLRD

Product Load Lock
*EXCLRD

Object Lock Each object in each of the product load's libraries is locked as follows:

Message queues are locked *EXCLRD.

The following object types are locked *SHRRD:
- Library
- User profile
- Device description
- Mode description
- Connection list
- Network interface description
- Class of service description
- Controller description
- Line description

All other object types are locked *SHRNUPD.

Required Parameter Group

Product option information
INPUT; CHAR(35)
The product loads to be packaged. For more information, see "Product Option Information Format."

Repackage
INPUT; CHAR(4)
Whether or not to package product loads that were already packaged.

- ***NO** Packages product loads that have not already been packaged.
- ***YES** Packages all specified product loads, regardless of whether they are already packaged.

Allow object change
INPUT; CHAR(5)
Whether to prevent changes to the product information in the object description of each object added to an object list. You can change the allow change by program attribute by using the allow change by program field of the Change Object Description (QLICOBJD) API. See the allow change by program field in the QLICOBJD API for more information.

- ***SAME** Does not change the allow change by program attribute of the objects being packaged.
- ***NO** Changes the allow change by program attribute of the objects being packaged so that the product information in the object description cannot be changed by the Change Object Description (QLICOBJD) 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.

Product Option Information Format

The following describes the product option information parameter. For a detailed description of the fields in this table, see "Field Descriptions" on page 54-20.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(4)	Product option
4	4	CHAR(7)	Product ID
11	B	CHAR(6)	Release level
17	11	CHAR(8)	Load ID
25	19	CHAR(10)	Reserved

Field Descriptions

Load ID. Specifies which loads to package. Load IDs are 4 characters in length; for example, 2924 is the load ID for an English National Language Version (NLV). The following special values are valid:

**CODEDFT* The default code load ID, 5001, is used.

**ALL* For the specified product ID, release level, and product option, all product loads for which a product load object exists are packaged. If only a code load exists, the code load is packaged.

Product ID. The product ID of the product option to be packaged.

Product option. The product option to be packaged. Use 0000 for the base option. Valid values are 0000 through 0099, where each character is a digit.

Release level. The version, release, and modification level of the product option to be packaged. The release level must be a valid special value, or it must be in the form of VxRyMz, where x and y are 0 through 9 and z is 0 through 9 or A through Z. The following special value is valid:

**ONLY* The release level is determined by searching the system for a product definition (*PRDDFN) for the specified product ID. The release level is taken from the product definition. This value is not valid if there are product definitions for two or more release levels of the product on the system.

Reserved. This field must contain blanks; otherwise, an error occurs.

Error Messages

CPF0CB2 E Product identifier &1 not valid.
CPF0CEA E Loads not packaged for product &1 release &2 option &3.
CPF0CEB E Product loads not packaged.
CPF0CEC E Code load has no folders.
CPF0CED E Reserved fields in parameter 1 are not blank.
CPF0CE0 E Message file &5 in library &6 not found.
CPF0CE1 E &1 not valid for repackage parameter.
CPF0CE3 E &9 product loads packaged, &10 product loads not packaged. See job log.
CPF0CE4 E Secondary language product load not packaged.
CPF0CE5 E &1 not valid for allow change parameter.
CPF0CE6 E Public authority for library &6 not valid for packaging.
CPF0CE7 E Product &1 release &2 not packaged.
CPF0CE8 E Primary folder list not correct.
CPF0CE9 E Development folder list not correct.
CPF0CFA E Load &4 for option &3 not in product definition.
CPF0CFB E Language load not packaged.
CPF0CFC E Product definition not found.
CPF0CFD E Code load not found for product &1 release &2 option &3.
CPF0CFE E Object &5 type *&7 in &6 not in development library &8.
CPF0CFF E Multiple releases available.
CPF0CF1 E Object &5 in &6 type *&7 associated with PTF &8.
CPF0CF4 E Object description not correct.
CPF0CF5 E Exit program &5 in library &6 not found.
CPF0CF6 E Packaging operation failed for &5 in &6 type *&7.
CPF0CF7 E Product &1 release &2 option &3 load &4 already packaged.
CPF0C4B E Product availability object &2/&1 recovery required.
CPF0C4C E Cannot allocate object &1 in library &2.
CPF0C4D E Error occurred while processing object &1 in library &2.
CPF0C8A E Product option &1 not valid.
CPF0C84 E Load identifier &4 not valid.
CPF2150 E Object information function failed.
CPF2151 E Operation failed for &2 in &1 type *&3.
CPF2225 E Not able to allocate internal system object.
CPF2352 E Program &1 in &2 received wrong parameters.
CPF24B4 E Severe error while addressing parameter list.
CPF2451 E Message queue &1 is allocated to another job.
CPF3CF1 E Error code parameter not valid.
CPF358A E Release not valid.
CPF7304 E File &1 in &2 not changed.
CPF8A06 E Document &2 or folder &3 partially created in folder &1.
CPF8A75 E Not authorized to access folder &1.
CPF8A77 E Folder &1 not found.
CPF8A78 E Folder &1 in use.
CPF8A79 E Folder &1 is logically damaged.
CPF8100 E All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9012 E Start of document interchange session not successful for &1.
CPF9801 E Object &2 in library &3 not found.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9804 E Object &2 in library &3 damaged.
CPF9806 E Cannot perform function for object &2 in library &3.
CPF9809 E Library &1 cannot be accessed.
CPF9810 E Library &1 not found.
CPF9811 E Program &1 in library &2 not found.
CPF9812 E File &1 in library &2 not found.
CPF9814 E Device &1 not found.
CPF9830 E Cannot assign library &1.
CPF9831 E Cannot assign device &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Release License (QLZARLS) API

Parameters

Required Parameter Group:

1	Product identification	Input	Char(*)
2	Product identification format name	Input	Char(8)
3	License user	Input	Char(*)
4	License user format name	Input	Char(8)
5	Error code	I/O	Char(*)

The Release License (QLZARLS) API is the opposite to the Request License API. Whatever was previously requested is now released and the usage count is decremented. When using this API, specify the same parameters as on the corresponding Request License API.

Required Parameter Group

Product identification
INPUT; CHAR(*)
Information that uniquely identifies the product or feature whose licensed use is to be released. The structure of this information is determined by the name of the format.

Product identification format name
INPUT; CHAR(8)
The name of the format that describes the product identification. The only format name supported is:

LICP0100 See "LICP0100 Format."

License user
INPUT; CHAR(*)
The name to which use of the license product is assigned and tracked. The structure of this information is determined by the content of the license user format name parameter.

License user format name
INPUT; CHAR(8)
The name of the format that describes the license user. The only format name supported is:

LICL0100 See "LICL0100 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.

LICP0100 Format

To release the licensed use of a product or feature, you must specify the same information that was specified on the Request License (QLZAREQ) API. The following table identifies the product or feature whose licensed use is to be released. For a detailed description of the fields in this table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(7)	Product ID
7	7	CHAR(6)	Release level
13	D	CHAR(4)	Feature

LICL0100 Format

To release the licensed use of a product or feature, you must specify the same information that was specified on the Request License (QLZAREQ) API. The following table identifies the license user whose licensed use is to be released. For a detailed description of the field in this table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	License user

Field Descriptions

Feature. The feature of the product. Valid values for the feature are 5001 through 9999.

License user. The name to which use of the license product is assigned and tracked. For registered use, specify the nonblank name you want assigned as the license user. For concurrent use, specify *JOB.

Product ID. The product ID of the product or feature whose licensed use is to be released.

Release level. The version, release, and modification level of the product or feature. The release level must be a valid special value, or the release level must be in the format VxRyMz. Valid values for x and y are 0 through 9, and valid values for z are 0 through 9 and A through Z. The valid special value is:

**ONLY* The release level is determined by searching the system for a given product. The release level is taken from the product definition. This value is not valid if more than one product definition exists for the same product ID.

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.
CPF9E1C E License user parameter not valid.
CPF9E11 E License information not retrieved.
CPF9E12 E License information not available.
CPF9E13 E More than one release found.
CPF9E15 E Error in license management function.

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

Request License (QLZAREQ) API

Parameters

Required Parameter Group:

1	Product identification	Input	Char(*)
2	Product identification format name	Input	Char(8)
3	License user	Input	Char(*)
4	License user format name	Input	Char(8)
5	Error code	I/O	Char(*)

The Request License (QLZAREQ) API requests the use of a product that has been packaged for licensed use. The request causes the usage count to be compared with the usage limit. The use is assigned to the name specified in the license user parameter. The use will remain assigned to the license user until it is released (see "Release License (QLZARLS) API" on page 54-20).

Required Parameter Group

Product identification
INPUT; CHAR(*)
Information that uniquely identifies the product or feature whose licensed use is requested. The structure of this information is determined by the name of the format.

Product identification format name
INPUT; CHAR(8)
The name of the format that describes the product identification. The only format name supported is:

LICP0100 This format consists of the product ID, release level, and code load ID. See "LICP0100 Format."

License user
INPUT; CHAR(*)
The name to which use of the license product is assigned and tracked. The structure of this information is determined by the content of the license user format name parameter.

License user format name
INPUT; CHAR(8)
The name of the format that describes the license user. The only format name supported is:

LICL0100 See "LICL0100 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.

LICP0100 Format

The following format identifies the product or feature whose use is requested. For a detailed description of the fields in this table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(7)	Product ID
7	7	CHAR(6)	Release level
13	D	CHAR(4)	Feature

LICL0100 Format

The following format identifies the product or feature whose use is requested. For a detailed description of the field in this table, see "Field Descriptions" on page 54-21.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	License user

Field Descriptions

Feature. The feature of the product. Valid values for the feature are 5001 through 9999.

License user. The name to which use of the license product is assigned and tracked. For registered use, specify the nonblank name you want assigned as the license user. For concurrent use, specify *JOB.

Product ID. The product ID of the product or feature whose licensed use is requested.

Release level. The version, release, and modification level of the product or feature. The release level must be a valid special value, or the release level must be in the format V*x*R*y*M*z*. Valid values for *x* and *y* are 0 through 9, and valid values for *z* are 0 through 9 and A through Z. The valid special value is:

**ONLY* The release level is determined by searching the system for a given product. The release level is taken from the product definition. This value is not valid if more than one product definition exists for the same product ID.

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

| CPF9E1C E License user parameter not valid.
| CPF9E11 E License information not retrieved.
| CPF9E12 E License information not available.
| CPF9E13 E More than one release found.
| CPF9E15 E Error in license management function.
| CPF9E17 E Usage limit exceeded for product &1, user added.
| CPF9E18 E Attempt made to exceed usage limit for product &1, user not added.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve License Information (QLZARTV) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name for receiver variable	Input	Char(8)
4	Product identification	Input	Char(*)
5	Product identification format name	Input	Char(8)
6	Error code	I/O	Char(*)

The Retrieve License Information (QLZARTV) API returns license information about a software product. The information retrieved consists of the following;

- Usage type
- Compliance type
- Usage limit
- License term
- Usage count

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable to receive the requested license information.

Length of receiver variable
INPUT; BINARY(4)
The length of the license information to be received. For example, the length of the LICR0100 format data is 39 bytes.

Format name for receiver variable
INPUT; CHAR(8)
The name of the format that identifies the type of license information to be retrieved. The only format name supported is:

LICR0100 Basic license information is retrieved. For more information, see "LICR0100 Format."

Product identification
INPUT; CHAR(*)
Information that uniquely identifies the product or feature whose license information will be retrieved. The structure of this information is determined by the name of the format.

Product identification format name
INPUT; CHAR(8)
The name of the format that describes the product identification. The only format name supported is:

LICP0100 See "LICP0100 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.

LICP0100 Format

The following table describes the format name supported for the format for product identification parameter. The format identifies the product or feature whose license information is to be retrieved. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 54-24.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(7)	Product ID
7	7	CHAR(6)	Release
13	D	CHAR(4)	Feature

LICR0100 Format

The following describes the format of the license information returned in the receiver variable parameter. For detailed descriptions of the fields in the table, see "Field Descriptions" on page 54-24.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Usage limit
4	4	BINARY(4)	Usage count
8	8	CHAR(2)	Usage type
10	A	CHAR(2)	Compliance type
12	C	CHAR(6)	License term
18	12	CHAR(6)	Release level

Field Descriptions

Compliance type. The compliance type associated with this license. The compliance type determines the action taken when the value of the usage limit field is exceeded. The valid values are:

01 The usage limit cannot be exceeded. The user who attempts to use the product or feature after the usage limit has been reached is prevented from accessing the product or feature until the appropriate action is taken to increase the usage limit. A message indicating an attempt was made to exceed the usage limit is sent to the QSYSOPR message queue and to the message queues specified on the Change License Information (CHGLICINF) command.

02 The user who attempts to use the product or feature after the usage limit has been reached is allowed access. A warning message indicating the usage limit has been exceeded is sent to QSYSOPR and to the message queues specified on the Change License Information (CHGLICINF) command.

Feature. The feature of the product. Valid values for the feature are 5001 through 9999.

License term. The extent of time the authorized usage limit for a product lasts. Each time a new license term is installed for a product, the authorized usage limit must be set by using the Work with License Information (WRKLICINF) command. Possible values are:

Vx The authorized usage limit is valid for the entire version of the product or feature.

VxRx The authorized usage limit is valid for the entire release of the product or feature.

VxRxMx The authorized usage limit is valid only for the modification level of the product.

Product ID. The product ID of the product or feature whose license information is to be retrieved.

Release. The version, release, and modification level of the product or feature as specified in the format for the product identification parameter. The release must be a valid special value, or the release must be in the format V*x*R*y*M*z*. Valid values for *x* and *y* are 0 through 9, and valid values for *z* are 0 through 9 and A through Z. The valid special value is:

*ONLY The release level is determined by searching the system for a given product. The release level is taken from the product definition. This value is not valid if more than one product definition exists for the same product ID.

Release level. The version, release, and modification level of the product whose license information was requested. This is returned in the receiver variable parameter. If you specified *ONLY in the release field of the LICP0100 format, the actual release level is returned here.

Usage count. The usage count for the product or feature at the time of the retrieve operation. Valid values are 0 through 999999.

Usage limit. The usage limit for this license.

-1 Any number of users are allowed to access the product or feature.

0–999999 The number of users allowed to access the product.

Usage type. The usage type associated with this license. The valid values are:

01 The usage type is concurrent. It is for the number of unique jobs accessing the product at one time.

02 The usage type is registered. It is for the number of unique license users registered by the product.

Error Messages

CPF2206 E User needs authority to do requested function.
CPF2207 E Not authorized to use object &1 in library &3 type &2.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF358A E Release not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.
CPF9E11 E License information not retrieved.
CPF9E13 E More than one release found.
CPF9E15 E Error in license management function.

Retrieve Product Information (QSZRTVPR) 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	Product information	Input	Char(27)
5	Error code	I/O	Char(*)

The Retrieve Product Information (QSZRTVPR) API returns information about a software product. The information is requested by specifying a product ID, release level, option number, and load ID; not by specifying an object name. The Display Software Resources (DSPSFWRSC) command will obtain a list of installed products about which you can retrieve information.

You can use this API to:

- Retrieve general information about a product load, including whether the product load is installed or not.

- Retrieve the library list of a product load.
- Retrieve the folder list of a product load.
- Retrieve the object list of a product load.
- Retrieve the list of option and load ID pairs that are valid for a product ID and release combination, based on what is listed in the product definition (*PRDDFN) for that product ID and release combination.
- Retrieve information from a product definition, including the copyright information, release date, message file name and library, whether the product allows multiple releases, the message ID for each option, and whether each option allows dynamic naming.
- Retrieve the current release level of the operating system.
- Retrieve the previous release level of the operating system.
- Retrieve a list of valid release levels of the operating system from a given release level through the currently installed release level.

Note: The Retrieve Object Description (QUSROBJD) API can be used to retrieve product information from the object description of an object. The product ID and release level from the object description is returned by QUSROBJD in format OBJD0300.

Authorities and Locks

Product Availability Authority
None

Product Availability Lock
*SHRRD

The product availability object resides in the QUSRSYS library.

Product Definition Authority
None

Product Load Authority
None

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable to receive the requested information.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable in bytes. The value specified must be at least 8.

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

PRDR0100
Returns basic information about the product load. For more information, see "PRDR0100 Format" on page 54-26.

PRDR0200
Returns a list of the principal and additional libraries for this product load, along with the basic information. Exit program names and other related information are also returned. For more information, see "PRDR0200 Format" on page 54-26.

PRDR0300
Returns a list of the folders for this product load, along with the basic information. For more information, see "PRDR0300 Format" on page 54-27.

PRDR0400
Returns a list of the packaged objects for this product load, along with the basic information. For more information, see "PRDR0400 Format" on page 54-27.

PRDR0500
Returns the information that was entered when the product definition object was created. This includes a record for each option listed in the product definition (*PRDDFN) for that product ID and release level. For more information, see "PRDR0500 Format" on page 54-28.

PRDR0600
Returns a list of option and load ID pairs that are valid for the specified product ID and release level, based on what is listed in the product definition (*PRDDFN) for that product ID and release. For more information, see "PRDR0600 Format" on page 54-28.

PRDR0700
Returns a list of release levels of the operating system. The list starts with the release level passed in by the caller and includes all releases of the operating system through the currently installed release. For more information, see "PRDR0700 Format" on page 54-28.

Product information
INPUT; CHAR(27)
The product ID, release level, product option, and load ID for which information is to be retrieved. For more information, see "Product Information 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.

Product Information Format

The format of the product information parameter is as follows. For detailed descriptions of each field, see "Field Descriptions" on page 54-26.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(7)	Product ID
7	7	CHAR(6)	Release level
13	D	CHAR(4)	Product option
17	11	CHAR(10)	Load ID

Field Descriptions

Load ID. The load ID for which information is being requested. Load IDs are 4 characters in length; for example, 2924 is the load ID for an English National Language Version (NLV). You can use this special value for the load ID:

*CODE The load ID of the code load for the given product ID, release level, and option is used.

Product ID. The product ID for which information is being requested. You can use this special value for the product ID:

*OPSYS The product ID for the operating system for the specified release level is used. The product ID depends upon the release level specified. For example, if V1R3M0 is specified for the release level, then product ID 5728SS1 is used. If V2R2M0 is specified for the release level, then product ID 5738SS1 is used.

Product option. The option number for which information is being requested. Use 0000 for the base option. Valid values are 0000 through 0099, where each character is a digit.

Release level. The release level for which information is being requested. The release level must be a valid special value, or the release level must be in the format VxRxMy, where valid values for x are 0 through 9, and valid values for y are 0 through 9 and A through Z. You can use these special values for the release level:

*CUR Uses the release level of the currently installed operating system.

*PRV Uses the previous release with modification level 0 of the operating system. Examples follow.

Example 1: If the current release level is V2R1M0, specifying *PRV for the release level parameter returns V1R3M0.

Example 2: If the current release level[1] is V2R1M1, specifying *PRV for the release level parameter returns V1R3M0, not V2R1M0.

Example 3: If the current release level[1] is V2R2M0, specifying *PRV for the release level parameter returns V2R1M0.

Format of the Returned Information

Information returned in the format name parameter can be in one of the following formats. For detailed descriptions of the fields for each format, see "Field Descriptions" on page 54-28.

PRDR0100 Format: If the product load is not known to the system, an error (CPF0C1F) will occur.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Reserved
12	C	CHAR(7)	Product ID
19	13	CHAR(6)	Release level
25	19	CHAR(4)	Product option
29	1D	CHAR(4)	Load ID
33	21	CHAR(10)	Load type
43	2B	CHAR(10)	Symbolic load state
53	35	CHAR(10)	Load error indicator
63	3F	CHAR(2)	Load state
65	41	CHAR(1)	Supported flag
66	42	CHAR(2)	Registration type
68	44	CHAR(14)	Registration value
82	52	CHAR(2)	Reserved
84	54	BINARY(4)	Offset to additional information
88	58	CHAR(*)	Reserved

PRDR0200 Format: If the *PRDLOD object does not exist, an error (CPF0C1F) will occur.

The fields following the library records field define that array. The number of entries in the array is the number of primary libraries for this load.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(*)	Everything from format PRDR0100

[1] In these examples, the current release level values are used to show how *PRV is determined; this API did not exist before V2R3M0.

Offset		Type	Field
Dec	Hex		
The decimal and hexadecimal offsets are determined by using the value in the offset to additional information field of format PRDR0100.		CHAR(10)	Secondary language library name
		CHAR(2)	Reserved
		BINARY(4)	Number of primary libraries
		BINARY(4)	Offset to library records
		CHAR(*)	Reserved
		ARRAY of CHAR(*)	Library records
The decimal and hexadecimal offsets are determined by using the value in the offset to library records field and the offset to next library record field.		BINARY(4)	Offset to next library record
		CHAR(10)	Primary library name
		CHAR(10)	Installed library name
		CHAR(10)	Library type
		CHAR(10)	Library authority
		CHAR(10)	Library create authority
		CHAR(10)	Postoperation exit program name
		BINARY(4)	Number of preoperation exit program names
		ARRAY of CHAR(10)	Preoperation exit program names
		CHAR(*)	Reserved

PRDR0300 Format: If the *PRDLOD object does not exist, an error (CPF0C1F) will occur.

The fields following the folder records field define an entry in that array. The number of entries in the array is the number of primary folders for this load.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(*)	Everything from format PRDR0100
The decimal and hexadecimal offsets are determined by using the value in the offset to additional information field of format PRDR0100.		BINARY(4)	Number of primary folders
		BINARY(4)	Length of one folder record
		BINARY(4)	Offset to folder records
		CHAR(*)	Reserved
		ARRAY of CHAR(*)	Folder records
The decimal and hexadecimal offsets are determined by using the value in the offset to folder records field and the length of one folder record field.		CHAR(63)	Primary folder
		CHAR(63)	Installed folder
		CHAR(*)	Reserved

PRDR0400 Format: If the product load has not been packaged, an error (CPF0C1F) will occur. If there has been PTF activity for this product load, this list might not contain all the objects that would be saved by the Save Licensed Program (SAVLICPGM) command. Error CPF0C1B is returned if this format is requested for the base option of the operating system.

The fields following the object records field define an entry in that array. The number of entries in the array is the number of objects packaged for this load.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(*)	Everything from format PRDR0100
The decimal and hexadecimal offsets are determined by using the value in the offset to additional information field of format PRDR0100.		BINARY(4)	Number of objects
		BINARY(4)	Length of one object record
		BINARY(4)	Offset to object records
		CHAR(*)	Reserved
		ARRAY of CHAR(*)	Object records
The decimal and hexadecimal offsets are determined by using the value in the offset to object records field and the length of one object record field.		CHAR(10)	Object name
		CHAR(10)	Installed library name
		CHAR(10)	Object type
		CHAR(*)	Reserved

PRDR0500 Format: If the product definition for the specified product ID and release level does not exist, an error (CPF0C1F) will occur. Error CPF0C1B is returned if this format is requested without specifying product option 0000 and load ID *CODE. Product option 0000 must be specified for this format even though the information returned is then for all options.

The fields following the option records field define an entry in that array. The number of entries in the array is the number of product options for this product and release.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(*)	Everything from format PRDR0100
The decimal and hexadecimal offsets are determined by using the value in the offset to additional information field of format PRDR0100.		CHAR(1)	Allow multiple releases
		CHAR(1)	Reserved
		CHAR(6)	Release date
		CHAR(4)	Copyright first year
		CHAR(4)	Copyright current year
		CHAR(10)	Message file object name
		CHAR(10)	Message file library name
		BINARY(4)	Number of option records
		BINARY(4)	Length of one option record
		BINARY(4)	Offset to option records
		CHAR(*)	Reserved
		ARRAY of CHAR(*)	Option records
The decimal and hexadecimal offsets are determined by using the value in the offset to option records field and the length of one option record field.		CHAR(4)	Product option
		CHAR(1)	Allow dynamic naming
		CHAR(7)	Product option message ID
		CHAR(*)	Reserved

PRDR0600 Format: Error CPF0C1F occurs if the product definition does not exist. Error CPF0C1B is returned if this format is requested without specifying product option 0000 and load ID *CODE.

The fields following the load records field define an entry in that array. The number of entries in the array is the number of loads for this product and release.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(*)	Everything from format PRDR0100
The decimal and hexadecimal offsets are determined by using the value in the offset to additional information field of format PRDR0100.		BINARY(4)	Number of load records
		BINARY(4)	Length of one load record
		BINARY(4)	Offset to load records
		CHAR(*)	Reserved
		ARRAY of CHAR(*)	Load records
The decimal and hexadecimal offsets are determined by using the value in the offset to load records field and the length of one load record field.		CHAR(4)	Product option
		CHAR(4)	Load ID
		CHAR(*)	Reserved

PRDR0700 Format: When this format is requested, valid values for the release level parameter are V1R3M0 and all release levels for which the operating system was made available, through the currently installed release level of the operating system.

If the release parameter is not a valid value, an error (CPF0C1C) will occur. Error CPF0C1B is returned if this format is requested without specifying the product option as 0000, product ID as *OPSYS, and load ID as *CODE.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
04	04	BINARY(4)	Bytes available
08	08	BINARY(4)	Number of releases returned
12	0C	CHAR(4)	Reserved
16	10	ARRAY of CHAR(6)	Release level (for each returned operating system release level)

Field Descriptions

Allow dynamic naming. Whether the names of product libraries and root folders for this product option can be dynamically changed without causing a product error. Possible values are:

0 Cannot be dynamically named.

| *1* Can be dynamically named.

| **Allow multiple releases**. Whether this product can be installed at a release level different from the current release level without installing over the current release. Possible values are:

| *0* This product cannot be installed at a release level different from the current release level without installing over the current release.
| *1* This product can be installed at a release level different from the current release level without installing over the current release.

| **Bytes available**. The number of bytes of data available to be returned to the user.

| **Bytes returned**. The number of bytes returned to the user. This is the lesser of the number of bytes available and the length of the receiver variable.

| **Copyright current year**. The value specified for the copyright current year when the product definition for this product load was created. If no copyright current year was specified when the product definition was created, the copyright current year is blank.

| **Copyright first year**. The value specified for the copyright first year when the product definition for this product load was created. If no copyright first year was specified when the product definition was created, the copyright first year is blank.

| **Folder records**. An array in which each entry includes the primary folder, installed folder, and reserved fields.

| **Installed folder**. The name of the folder specified as the development folder when the load was created, or the name given to the primary folder when the product was installed.

| **Installed library name**. For a library record, the name of the library specified as the development library when the load was created, or the name given to the primary library when the product was installed.

| For an object record, the name of the library where the object should exist. The object might not exist in this library if the object had been deleted, or the library had been renamed.

| **Length of one folder record**. The number of bytes in each folder record.

| **Length of one load record**. The number of bytes in each load record.

| **Length of one object record**. The number of bytes in each object record.

| **Length of one option record**. The number of bytes in each option record.

| **Library authority**. The public authority given to the library by the Restore Licensed Program (RSTLICPGM) command when this load is installed if the library does not exist. This field will be blank if the product load has not been successfully packaged. Possible values are:

| **ALL* The library is created with the public authority set to *ALL.
| **USE* The library is created with the public authority set to *USE.
| **CHANGE* The library is created with the public authority set to *CHANGE.
| **EXCLUDE* The library is created with the public authority set to *EXCLUDE.

| **Library create authority**. The create authority set for this library by the Restore Licensed Program (RSTLICPGM) command when this load is installed if the library does not exist. This field will be blank if the product load has not been successfully packaged. Possible values are:

| **ALL* The library is created with the create authority set to *ALL.
| **USE* The library is created with the create authority set to *USE.
| **CHANGE* The library is created with the create authority set to *CHANGE.
| **EXCLUDE* The library is created with the create authority set to *EXCLUDE.

| **Library records**. An array in which each entry includes the following fields:

| Offset to next library record
| Primary library name
| Installed library
| Library type
| Library authority
| Library create authority
| Postoperation exit program name
| Number of preoperation exit program names
| Preoperation exit program names
| Reserved

| **Library type**. The type of library created by the Restore Licensed Program (RSTLICPGM) command when this load is installed if the library does not exist. This field will be blank if the product load has not been successfully packaged. Possible values are:

| **PROD* The library created is a production library.
| **TEST* The library created is a test library.

| **Load error indicator**. Whether there is a known error for this load. The possible values are:

| **ERROR* An error was found the last time that the state of this load was checked or updated. For example, a restore, delete, or save licensed program function might be in progress or might not have completed. The state of a load can be checked using the Check Product Option (CHKPRDOPT)

command. See the *CL Reference* for information on the CHKPRDOPT command.

*NONE — No error was found the last time that the state of this load was checked or updated.

Note: This does not mean that the product is necessarily installed. Refer to the symbolic load state field to determine if the load is installed or not.

Load ID. The load ID of the product load for which information was returned. For the load records, the load ID field returns the load IDs that have been specified when the product definition was created.

Load records. An array for which each entry includes the product option, load ID, and a reserved field.

Load state. The state of the load for which information was returned. The possible values are:

10 — The load is defined. The product load object for this load does not exist. When a product definition is created, a code load is defined for each product option, and language loads can be defined.

20 — The product load object for this load exists. It must be packaged with the Package Product Option (PKGPRDOPT) command or the Package Product Option (QSZPKGPO) API before it can be saved using the Save Licensed Program (SAVLICPGM) command.

3E — A Restore Licensed Program (RSTLICPGM) command did not complete successfully. A preoperation exit program failed. The product being replaced had been packaged, but not installed.

3F — A RSTLICPGM command failed. A preoperation exit program did not fail. The product being replaced had been packaged, but not installed.

30 — The product load object for this load has been packaged with the PKGPRDOPT command or the QSZPKGPO API.

32 — The product load object for this load has been packaged with the PKGPRDOPT command or the QSZPKGPO API. One of the following occurred:

- A development library or folder was renamed, but the product does not allow dynamic naming. (A product specifies whether or not it allows dynamic naming when the product definition object is created.)
- The product definition or product load for a packaged load was renamed or moved to another library.

33 — The product load object for this load has been packaged with the PKGPRDOPT command or the QSZPKGPO API. However, an object was found to be damaged the last time that the CHKPRDOPT command or SAVLICPGM command was used for this load.

34 — The product load object for this load has been packaged with the PKGPRDOPT command or the QSZPKGPO API. One of the following occurred:

- An attempt was made to delete the product load using the delete licensed program function and the function failed.
- A packaged object was missing the last time that the CHKPRDOPT command or SAVLICPGM command was used for this load.

35 — A RSTLICPGM command is in progress. The product being replaced had been packaged, but not installed.

38 — A Delete Licensed Program (DLTLICPGM) command is in progress. The product being deleted had been packaged, but not installed.

50 — A RSTLICPGM command is in progress. The product being replaced had been installed.

53 — A DLTLICPGM command is in progress. The product being deleted had been installed.

59 — This product is an IBM-supplied product, and it is not compatible with the currently installed release level of the operating system. An error occurred when the product was restored or when the operating system was installed. The IBM-supplied product is at a release level earlier than V2R2M0, which is not supported by the SAVLICPGM command.

6E — A RSTLICPGM command did not complete successfully. A preoperation exit program failed. The product being replaced had been installed.

6F — A RSTLICPGM command failed. The failure was not a preoperation exit program or postoperation exit program. The product being replaced had been installed.

60 — The product load (*PRDLOD) object for this load was loaded onto the system by the RSTLICPGM command.

61 — The product load (*PRDLOD) object for this load was loaded onto the system by the RSTLICPGM command, but a postoperation exit program failed.

62 — An installed library or folder was renamed, but the product does not allow dynamic naming. (A product specifies whether or not it allows dynamic naming when the product definition object is created.)

63 — The product load (*PRDLOD) object for this load was installed by the RSTLICPGM command, but an object is damaged.

64 — The product load (*PRDLOD) object for this load was installed by the RSTLICPGM command, but one of the following occurred:

- An object was found to be missing when the CHKPRDOPT command or the SAVLICPGM command was used.
- An error occurred while the DLTLICPGM command was being used.

67 — The CHKPRDOPT command was used for this product load, but the postoperation exit program failed or indicated that an error was found.

90 — The product load was installed successfully. If an object was missing or was damaged, and the problem was corrected, using the CHKPRDOPT command sets the state back to 90.

Load type. The type of load for which information was returned. The possible values are:

| *CODE The load is a code load.
| *LNG The load is a language load.

| **Message file library name**. The name of the library for the message file that contains the messages describing the product and its options.

| **Message file object name**. The name of the message file that contains the messages describing the product and its options.

| **Number of load records**. The number of loads for this product option. If the receiver variable was not large enough to hold all the load records, this number may be larger than the number of load records actually returned.

| **Number of objects**. The number of packaged objects for this load. If the receiver variable was not large enough to hold all the objects, this number may be larger than the number of objects actually returned.

| **Number of option records**. The number of options for this product and release level. If the receiver variable was not large enough to hold all the option records, this number may be larger than the number of option records actually returned.

| **Number of preoperation exit program names**. The number of preoperation exit programs for this load for this primary library. If there are no preoperation exit programs for this library, this will be 0.

| **Number of primary folders**. The number of primary folders for this load. If the receiver variable was not large enough to hold all the folder records, this number may be larger than the number of folder records actually returned.

| **Number of primary libraries**. The number of primary libraries for this load. If the receiver variable was not large enough to hold all the library information, this number may be larger than the number of libraries actually returned. The first record contains the principal primary library information. Subsequent records contain the information for the additional libraries, if the load has any additional libraries.

| **Number of releases returned**. The number of release levels returned for format PRDR0700.

| **Object name**. The name of an object for this load.

| **Object records**. The objects in the object record are ordered by library. All objects for the principal library are first. Within each library, the objects are ordered by object type, but with product loads first and product definitions second, followed by all other object types. The record has the following fields:

| Object name
| Installed library name
| Object type
| Reserved

| **Object type**. The symbolic object type of the object.

| **Offset to additional information**. Offset from the beginning of the receiver variable to the start of the rest of the information for a given format. This is to allow for expansion of the basic information. For format PRDR0100, this is 0.

| **Offset to folder records**. Offset from the beginning of the receiver variable to the start of the first folder record for format PRDR0300. This is to allow for expansion of the basic folder information.

| **Offset to library records**. Offset from the beginning of the receiver variable to the start of the first library record for format PRDR0200. This is to allow for expansion of the basic library information.

| **Offset to load records**. Offset from the beginning of the receiver variable to the start of the first load record for format PRDR0600. This is to allow for expansion of the basic load information.

| **Offset to next library record**. Offset from the beginning of the receiver variable to the start of the next library record for format PRDR0200. If there are no more library records, then this is 0.

| **Offset to object records**. Offset from the beginning of the receiver variable to the start of the first object record for format PRDR0400. This is to allow for expansion of the basic object information.

| **Offset to option records**. Offset from the beginning of the receiver variable to the start of the first option record for format PRDR0500. This is to allow for expansion of the basic option information.

| **Option records**. An array for which each entry includes the following fields:

| Product option
| Allow dynamic naming
| Product option message ID
| Reserved

| **Postoperation exit program name**. The name of the postoperation exit program for this load for this primary library. If there is no postoperation exit program for this library, this will be blank.

| **Preoperation exit program names**. An array of the preoperation exit programs for this load for this primary library. If there are no preoperation exit programs for this library, this will be an array of length 0.

| **Primary folder**. The name of a primary folder for this load.

| **Primary library name**. The name of the primary library that was specified when the product load object was created.

| **Product ID**. The product ID for which information was returned.

Product option. The product option for which information was returned.

Product option message ID. The message ID associated with this product option. The message ID was specified when the product definition was created.

Registration type. The registration type associated with the product. The registration type and registration value together make up the registration ID for the product. The possible values are:

02 Registration type *PHONE was specified when the product load or product definition was created.
04 The registration value is the same as the registration value for OS/400.
08 Registration type *CUSTOMER was specified when the product load or product definition was created.

Registration value. The registration value associated with the product. The registration type and registration value together make up the registration ID for the product.

Release date. Indicates the value specified for the release date when the product definition for this product load was created. The release date is in the format *yymmdd*, where *yy* equals year, *mm* equals month, and *dd* equals day. If no release date was specified when the product definition was created, then the release date is blank.

Release level. The release level of the product for which information was returned. For V2R3M0, when format PRDR0700 is requested, the valid values for this field are *CUR, *PRV, V1R3M0, V2R1M0, V2R1M1, V2R2M0, and V2R3M0.

Release level (for each returned operating system release level). The individual release level returned. One or more may be returned.

Reserved. An ignored field.

Secondary language library name. The secondary language library name that was specified when the load was created. If this is a code load, the secondary language library name is blank.

Supported flag. Whether this load is currently supported. A load can be supported by using the Work with Supported Products (WRKSPTPRD) command in the SystemView System Manager/400 licensed program. The possible values are:

0 The load is not supported.
1 The load is supported.

Symbolic load state. The symbolic state of the load for which information was returned. This value, in conjunction with the load error indicator, can be used to determine if the load is installed correctly. The possible values are:

**DEFINED*
The load is defined. The product load object for this load does not exist. When a product definition is created, a code load is defined for each product option, and language loads can be defined.
**CREATED*
The product load object for this load exists. It must be packaged with the PKGPRDOPT command before it can be saved using the SAVLICPGM command.
**PACKAGED*
The product load object for this load has been packaged with the PKGPRDOPT command.
**DAMAGED*
If this is for an option other than the base option or for a language load for the base option, the product load object has been damaged. If this is for the code load for the base option, the product definition for this product ID and release level has been damaged or the product load object has been damaged.
**LOADED*
Indicates one of the following:
- A restore licensed program function is in progress.
- A delete licensed program function is in progress.
- The product was created previous to V2R2M0, and there was an error during the process of converting the product information.

**INSTALLED*
The product load (*PRDLOD) object for this load was loaded onto the system by the RSTLICPGM command.

Error Messages

CPF0C1B E Requirements between parameters not satisfied.
CPF0C1C E Release level &1 not valid.
CPF0C1D E Load ID &1 not valid.
CPF0C1E E Error occurred during running of &1 API.
CPF0C1F E Product information not found.
CPF0C4B E Product availability object &2/&1 recovery required.
CPF0C4C E Cannot allocate object &1 in library &2.
CPF0C4D E Error occurred while processing object &1 in library &2.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF8191 E Product definition &4 in &9 damaged.
CPF8193 E Product load object &4 in &9 damaged.
CPF9838 E User profile storage limit exceeded.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Program Temporary Fix Information (QPZRTVFX) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	PTF information	Input	Char(50)
4	Format name	Input	Char(8)
5	Error code	I/O	Char(*)

The Retrieve PTF Information (QPZRTVFX) API returns information about a specific PTF. The information returned is determined by the format specified.

You can use the QPZRTVFX API to:

- Retrieve basic information about a PTF.
- Retrieve the cover letter member names for a PTF.
- Retrieve the prerequisites for a PTF.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The receiver variable that is to receive the information requested. The length of this area must be passed in the length of receiver variable parameter. The API returns only the data the area can hold.

Length of receiver variable
INPUT; Binary(4)
The length of the receiver variable. You can specify a smaller area than the format requested as long as you specify the length of receiver variable parameter correctly. If the length specified is larger than the size of the receiver variable, the results are not predictable. This value must be greater than or equal to 8.

PTF information
INPUT; CHAR(50)
The attributes of the PTF for which information is being requested. For more information on this parameter see "Format of PTF Information."

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

PTFR0100 Basic information; for details, see "PTFR0100 Format."

PTFR0200 This format contains the basic information as well as an array of cover letters in the format given. A maximum of 29 cover letters could be returned in the array. If the receiver variable does not have enough space for all the cover letters to be returned, only the cover letters that fit in the space provided will be returned. For details, see "PTFR0200 Format" on page 54-34.

PTFR0300 This contains the basic information plus an array containing information about the prerequisite PTFs. If the receiver variable does not have enough space to return all prerequisite information, only the information that will fit in the space provided will be returned. For details, see "PTFR0300 Format" on page 54-34.

Error code
I/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 PTF Information

For detailed descriptions of each field, see the "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	CHAR(7)	PTF ID
7	7	CHAR(7)	Product ID
14	E	CHAR(6)	Release level
20	14	CHAR(30)	Reserved

Field Descriptions

Product ID. The product ID for the PTF. The possible values are:

**ONLY* The product ID is not known but only one PTF exists on the system by this name.
product ID The product ID for the PTF.

PTF ID. The identifier of the PTF for which information is requested.

Release level. The release of the PTF for which information is requested. This field is ignored if *ONLY is specified in the product ID field. The format is:

VxRyMz The release of the PTF in the format VxRyMz, where x and y are any number between 0 and 9 and z is a number between 0 and 9 or a character between A and Z.

Reserved. This field must contain blanks.

PTFR0100 Format

This format returns basic information about the PTF. For detailed descriptions of each field, see the "Field Descriptions" on page 54-34.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Offset to additional information
12	C	CHAR(7)	Product ID
19	13	CHAR(7)	PTF ID
26	1A	CHAR(6)	Release level
32	20	CHAR(4)	Product option
36	24	CHAR(4)	Load ID
40	28	CHAR(1)	Loaded status
41	29	CHAR(1)	Cover letter status
42	2A	CHAR(1)	On-order status
43	2B	CHAR(1)	Save file status
44	2C	CHAR(10)	File name
54	36	CHAR(10)	File library name
64	40	CHAR(1)	PTF type
65	41	CHAR(*)	Reserved

PTFR0200 Format

The fields that follow the cover letter records field define an entry in that array. That group of fields is repeated by the number of different NLVs available for the cover letter field. For detailed descriptions of each field, see the "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0		All information from format PTFR0100
Note: The decimal and hexadecimal offsets are reached by using the offset to additional information field in format PTFR0100.			
		BINARY(4)	Offset to first cover letter record from the beginning
		BINARY(4)	Number of different NLVs available for the cover letter
		BINARY(4)	Length of cover letter record
		CHAR(*)	Reserved
		ARRAY of CHAR(*)	Cover letter records
		CHAR(4)	Cover letter national language version (NLV)
		CHAR(10)	Cover letter file name
		CHAR(10)	Cover letter library name
		CHAR(10)	Cover letter member name
		CHAR(*)	Reserved

PTFR0300 Format

The fields following the prerequisite record field define an entry in that array. That group of fields is repeated by the number of prerequisites. For detailed descriptions of each field, see the "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0		All information from format PTFR0100
Note: The decimal and hexadecimal offsets are reached by using the offset to additional information field in format PTFR0100.			
		BINARY(4)	Offset to first prerequisite record
		BINARY(4)	Number of prerequisites
		BINARY(4)	Length of prerequisite record
		CHAR(*)	Reserved
		ARRAY of CHAR(*)	Prerequisite record
		CHAR(7)	Prerequisite product ID
		CHAR(7)	Prerequisite PTF ID
		CHAR(6)	Release of prerequisite
		CHAR(*)	Reserved

Field Descriptions

Bytes available. The number of bytes of data available to be returned to the user.

Bytes returned. The number of bytes that were returned to the user. This is the lesser of the number of bytes available to be returned or the length of the receiver variable.

Cover letter file name. The file containing the cover letter member.

Cover letter library name. The library containing the cover letter file.

Cover letter member name. The member containing the cover letter.

Cover letter national language version (NLV). The NLV corresponds to a code that indicates what language the cover letter was written in. For more information on NLVs, see the *National Language Support Planning Guide*.

Cover letter records. The record containing the cover letter information.

Cover letter status. Whether a cover letter exists for the PTF. The following values are valid.

| *0* The PTF has no cover letter.
| *1* The PTF has a cover letter.

| **File library name**. The name of the library where the save file for the PTF is located. If no save file name has been reserved, this field will be blank.

| **File name**. The name of the file where the save file for the PTF is located. If no save file name has been reserved, this field will be blank.

| **Length of cover letter record**. The length of each cover letter record.

| **Length of prerequisite record**. The length of each prerequisite record.

| **Loaded status**. The current loaded status of the PTF. A PTF can have any of the following statuses:

| *0* The PTF has never been loaded.
| *1* The PTF has been loaded.
| *2* The PTF has been applied.
| *3* The PTF has been applied permanently.
| *4* The PTF has been permanently removed.
| *5* The PTF is damaged. An error occurred while applying the PTF. It needs to be reloaded and applied.
| *6* The PTF is superseded. Another PTF with a more recent fix for the problem has been received on the system.

| **Note:** These fields are returned as numbers instead of text because statuses are translatable text instead of special values. The text message that contains these values is CPX3501.

| This field may be blank.

| **Load ID**. The load ID of the product option for the PTF.

| **Number of different NLVs available for the cover letter**. Cover letter member names are returned in an array. This number indicates how many NLVs were available for the cover letter. If the length of the receiver variable was not large enough to hold all the cover letter NLVs available, the number of cover letter member names returned may be less than this value.

| **Number of prerequisites**. The number of prerequisite PTFs available. If the receiver variable was not large enough to hold all the prerequisite PTF records, the number of prerequisites returned may be less than this value.

| **Offset to additional information**. The offset from the beginning of the receiver variable to the start of either the cover letter information or the prerequisite information. This is to allow expansion of the basic information.

| **Offset to first cover letter record from the beginning**. The offset from the beginning of the receiver variable to the first cover letter record.

| **Offset to first prerequisites record**. The offset from the beginning of the receiver variable to the start of the first prerequisite record.

| **On-order status**. Whether the PTF has been ordered. The following values are valid:

| *0* The PTF has not been ordered or has already been received.
| *1* The PTF has been ordered.

| **Prerequisite product ID**. The product ID of the prerequisite PTF.

| **Prerequisite PTF ID**. The PTF ID of the prerequisite PTF.

| **Prerequisite record**. The information about the prerequisite. The record is an array containing the following fields.

| Prerequisite product ID
| Prerequisite PTF ID
| Release of prerequisite

| **Product ID**. The PTF is for this product. The product must be installed or supported.

| **Product option**. The PTF is for this option of the product.

| **PTF ID**. The identifier of the PTF.

| **PTF type**. The type of PTF. The possible values are:

| *0* PTF is delayed. The PTF must be applied at IPL time.
| *1* The PTF is immediate. The PTF can be applied immediately. No IPL is needed.
| *Blank* The PTF type is not known.

| **Release level**. The release of the PTF. It must be in the format of VxRyMz, where x and y are any number between 0 and 9 and z is any number between 0 and 9 or a character between A and Z.

| **Release of prerequisite**. The release of the prerequisite PTF.

| **Reserved**. This field is ignored.

| **Save file status**. Whether a save file exists for the PTF. This field should always be checked to determine if a save file exists. The following values are valid.

| *0* The PTF has no save file.
| *1* The PTF has a save file.

| Error Messages

| CPF0CB3 E Value for reserved field not valid.
| 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.
| CPF3C24 E Length of the receiver variable is not valid.

| CPF35BE E Product &1 &3 not supported or installed.
| CPF358A E Release not valid.
| CPF3902 E PTF &1 located more than once.
| CPF6602 E PTF &1-&2 &3 not found.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

Chapter 55. Software Product Exit Programs

You may write exit programs that are called by programs and by program temporary fixes (PTFs).

The software product exit programs follow:

Software Product Functions exit program is specified when creating products that will be restored, deleted, saved, and checked with CL commands.

Program Temporary Fix exit program is called when a PTF is temporarily or permanently applied or removed with the Apply PTF (APYPTF) or Remove PTF (RMVPTF) commands.

QLPUSER exit program is called during the automatic installation process and can be used by central sites when they are distributing products to remote locations.

Software Product Functions Exit Program

This exit program is specified when you create products that will be restored, saved, deleted, and checked with the Restore Licensed Program (RSTLICPGM), Delete Licensed Program (DLTLICPGM), Save Licensed Program (SAVLICPGM), and Check Product Option (CHKPRDOPT) commands. More information about creating your own product and using exit programs is available in the *SystemView* System Manager/400 User's Guide*.

Parameters

Required Parameter Group:

1	Function being performed	Input	Char(10)
2	Before or after indicator	Input	Char(10)
3	Language ID	Input	Char(4)
4	Library that contains exit programs	Input	Char(10)
5	Library where product currently exists	Input	Char(10)
6	Library specified by creator of product	Input	Char(10)
7	Library where product is going to exist	Input	Char(10)
8	Root folder that product currently uses	Input	Char(12)
9	Root folder specified by creator	Input	Char(12)
10	Root folder that product will use	Input	Char(12)
11	Release level of product	Input	Char(6)
12	Release level of product being restored	Input	Char(6)
13	Array of missing objects and their symbolic object types	Input	Array of Char(20)
14	Number of missing objects	Input	Binary(4)
15	Array of missing folders	Input	Char(63)
16	Number of missing folders	Input	Binary(4)

Required Parameter Group

Exit programs must be written to accept the following parameters:

Function being performed
INPUT; CHAR(10)
The function being performed when this exit program is called. The values are:

***RSTCODE (restore code)**
Restore program objects when the RSTLICPGM command runs.

***SAVCODE (save code)**
Save program objects when the SAVLICPGM command runs.

***DLTCODE (delete code)**
Delete program objects when the DLTLICPGM command runs.

***CHKCODE (check code)**
Check program objects when the CHKPRDOPT command runs.

***RSTLNG (restore language)**
Restore language objects when the RSTLICPGM command runs.

***SAVLNG (save language)**
Save language objects when the SAVLICPGM command runs.

***DLTLNG (delete language)**
Delete language objects when the DLTLICPGM command runs.

***CHKLNG (check language)**
Check language objects when the CHKPRDOPT command runs.

Before or after indicator
INPUT; CHAR(10)
Whether the exit program is being called before or after the actual operation (restore process, save process, and so on). The values are:

***BEFORE** The program is called before the actual operation.
***AFTER** The program is called after the actual operation.

Language ID
INPUT; CHAR(4)
The 4-digit feature code identifier. For the *RSTCODE, *SAVCODE, *DLTCODE, and *CHKCODE values of the function being performed parameter, this value will be blank because it only applies to the language functions.

Library that contains exit programs
INPUT; CHAR(10)
The library where exit programs for the product are located while the current function is being performed. You cannot assume this value will be either the library where the product currently exists or will exist.

Library where product currently exists
INPUT; CHAR(10)
If the product exists on the system when the exit program is called, this is the library that contains the product. Otherwise, this value is blank.

Library specified by creator of product
INPUT; CHAR(10)
The library that was specified on the Create Product Load (CRTPRDLOD) command or the Create Product Load (QSZCRTPL) API when you created the product.

Library where product is going to exist
INPUT; CHAR(10)
The library of the product after the product is restored. This value only applies when the function being performed is *RSTCODE and *RSTLNG. For other functions, this value is blank.

Root folder that product currently uses
INPUT; CHAR(12)
The root folder specified if the product is installed and has folders. It is only specified when calling the exit program for the principal library. Otherwise, this value is blank.

Root folder specified by creator
INPUT; CHAR(12)
The root folder that was specified on the CRTPRDLOD command or the QSZCRTPL API when the product was created. This only applies if the product has folders and is only specified when calling the exit program for the principal library.

Root folder that product will use
INPUT; CHAR(12)
The root folder specified only if the product is being restored and has folders. It is only specified when calling the exit program for the principal library. Otherwise, this value is blank.

Release level of product
INPUT; CHAR(6)
The version, release, and modification level of the product being restored, saved, deleted, or checked in the form VxRxMx.

Release level of product being restored
INPUT; CHAR(6)
The version, release, and modification level of the product being restored in the form VxRxMx. It is only specified if the product is being restored; otherwise, the value is blank.

Array of missing objects and their symbolic object types
INPUT; ARRAY of CHAR(20)
The list passed when the function being performed is *SAVCODE, *SAVLNG, *CHKCODE, or *CHKLNG. For *SAVCODE and *SAVLNG, the list will only be passed before the actual save operation. For *CHKCODE and *CHKLNG, the list will be passed after the check operation. The first 10 characters contain the object name, and the second 10 characters contain the object type.

Number of missing objects
INPUT; BINARY(4)
The number of missing objects in the previous array. This number is 0 when there are no objects in the list or when the function being performed is not *SAVCODE, *SAVLNG, *CHKCODE, or *CHKLNG. The maximum number is 499.

Array of missing folders
INPUT; CHAR(63) for each folder name
The list passed when the function being performed is *SAVCODE, *SAVLNG, *CHKCODE, or *CHKLNG. For *SAVCODE and *SAVLNG, the list will only be passed before the actual save operation. For *CHKCODE and *CHKLNG, the list will be passed after the check operation. This only applies if the product has folders and is only specified when calling the exit program for the principal library.

Number of missing folders
INPUT; BINARY(4)
The number of missing folders in the previous array. This number is 0 when there are no folders in the list or when the function being performed is not *SAVCODE, *SAVLNG, *CHKCODE, or *CHKLNG.

Error Messages

CPF3D95	Exit program processing failed.
CPF3D98	Exit program processing found error in product.
CPD3DC8	&5 &6 in &7 not found for product &1 option &2 release &4.
CPD3DE7	Folder &6 not found for product &1 option &2 release &4.

Program Temporary Fix Exit Program

Parameters

Required Parameter Group:

1	Product ID	Input	Char(7)
2	PTF ID	Input	Char(7)
3	Product release level	Input	Char(6)
4	Product option ID	Input	Char(4)
5	Product load ID	Input	Char(4)
6	PTF library	Input	Char(10)
7	User data	Input	Char(50)
8	Current PTF status	Input	Char(1)
9	PTF operation	Input	Char(1)

When a program temporary fix (PTF) is created, one or more PTF exit programs may be specified. The PTF exit programs are called when a PTF is temporarily applied, permanently applied, temporarily removed, or permanently removed. Exit programs are called at the end of the apply or remove processing. The run option field of the exit programs parameter determines when the exit program is called. (Refer to "Exit Programs Format" on page 54-12.) Exit programs eliminate the need for you to manually carry out special instructions to install the PTF.

Shipping the same exit program in two PTFs for the same product causes one PTF to supersede the other. Avoid this by including the PTF exit program when the product is initially packaged. If a PTF exit program already exists in the product, then specify *OBJLIST for the value of the exit program type field (see "Exit Programs Format" on page 54-12). Another way to avoid unwanted superseding PTFs is to ship the exit program with a different temporary object name in each PTF. When the exit program is being shipped with the PTF, specify the value *PTF for the exit program type field (see "Exit Programs Format" on page 54-12).

The interface between the Apply PTF (APYPTF) command or Remove PTF (RMVPTF) command and the exit program follows. Your exit program must have the following input parameters.

Product ID
INPUT; CHAR(7)
The software product that the PTF affects.

PTF ID
INPUT; CHAR(7)
The identification number of the PTF.

Product release level
INPUT; CHAR(6)
The release of the software product that the PTF is for in the format VxRyMz. Valid values for x and y are 0 through 9, and valid values for z are 0 through 9 and A through Z.

Product option ID
INPUT; CHAR(4)
The option of the software product that the PTF is for.

Product load ID
INPUT; CHAR(4)
The load of the software product that the PTF is for.

PTF library
INPUT; CHAR(10)
The name of the library in which the PTF objects were placed.

User data
INPUT; CHAR(50)
Any user data specified by the PTF originator to be passed to the exit program.

Current PTF status
INPUT; CHAR(1)
The current status of the PTF. The status is passed as encoded data.

0 Loaded but not applied
1 Applied temporarily

PTF operation
INPUT; CHAR(1)

The change being made to the status of the PTF is passed as encoded data.

0 Remove temporarily
1 Apply temporarily
2 Apply permanently
3 Remove permanently

Error Messages

CPF3638 Exit program failed, but changes were backed out.
CPF3639 Exit program failed and permanent changes were made.

If one of these messages is sent, the apply PTF process fails and no following PTFs will be applied or removed. This means that if you are installing a cumulative PTF package when this failure occurs, the cumulative package will not be applied.

If an exit program sends escape message CPF3638, any other exit programs already called for that PTF are called again. This allows changes to be backed out. During the back-out operation, if all the exit programs called by the PTF run successfully, the PTF status does not change from what it was before the PTF operation was attempted. For example, if the PTF was not applied and the PTF exit program failed while the PTF was being temporarily applied, then the status of the PTF would remain not applied. This message should only be issued if any changes made by the exit program were backed out.

When message CPF3639 is signaled, the PTF is marked damaged. This indicates that the PTF has been partially applied. Any objects contained in the PTF have already been replaced. Some of the exit programs may have completed successfully.

If an exit program signals any other escape messages, unpredictable results can be expected. Exit programs must monitor for all exceptions. Other error messages should be left in the job log for problem determination or should be sent as informational or diagnostic messages. Message CPF3569 can be used to indicate that a special handling program failed.

You can also send a completion message (CPC1214) if the exit program runs successfully.

For more information, see "Creating a Program Temporary Fix Exit Program" on page A-52.

QLPUSER Exit Program

The QLPUSER exit program is called during the automatic installation process. It can be used to restore products that are not listed on the Licensed Program menu. More information about the QLPUSER program is available in the *Central Site Distribution Guide*.

Parameters

Required Parameter:

1	Tape device name	Input	Char(10)

Required Parameter

Tape device name
INPUT; CHAR(10)
The tape device name that identifies the tape device to be used.

QLPUSER Exit Program Example

The following example is a control language (CL) program that sends an instruction to the operator at the target site system, restores a library containing an application, and copies the command to start the application in the QGPL library.

Note: The &DEVICE parameter in the example is the name of your tape device.

```
PGM PARM(&DEVICE)
SNDUSRMSG MSG('Load the tape containing Application 1, then press Enter.')
DCL VAR(&DEVICE) TYPE(*CHAR) LEN(10)
RSTLIB SAVLIB(APP1) DEV(&DEVICE)
CRTDUPOBJ OBJ(STRAPP1) FROMLIB(APP1) OBJTYPE(*CMD) TOLIB(QGPL)
ENDPGM
```

Part 21. Spooled File and Print APIs

Chapter 56. Spooled File and Print APIs

This section contains information about:

- Spooled file APIs
- Print APIs
- Exit program for customizing a separator page

The spooled file and print APIs are presented alphabetically and are followed by the exit program.

Spooled File APIs

Spooled file APIs obtain specific information about spooled files. For example, spooled file APIs can:

- Return a list of spooled files based on given selection criteria, such as a user or an output queue.
- Provide functions to access a specific spooled file from which the API can return the attributes and data of a spooled file or create a duplicate of a specific spooled file.

Spooled file APIs are useful in writing applications to clean up, save, and restore spooled files. Spooled file APIs include the following:

Close Spooled File (QSPCLOSP) closes a spooled file opened by the Open Spooled File (QSPOPNSP) API or created by the Create Spooled File (QSPCRTSP) API.

Create Spooled File (QSPCRTSP) creates a spooled file. The attributes for the spooled file are based on the values taken from the spooled file attributes parameter. When this spooled file is created, it does not contain any data.

Get Spooled File Data (QSPGETSP) gets data from an existing spooled file. The existing spooled file must have been opened by the Open Spooled File (QSPOPNSP) API. Data is retrieved from the existing spooled file by buffers (one or more) and stored in a user space. The data in the user space is used as source to the Put Spooled File Data (QSPPUTSP) API, which puts the data in the newly created spooled file.

List Spooled Files (QUSLSPL) generates a list of spooled files on the system into a user space. The list can include all spooled files or those of specific users, output queues, form types, or user-specified data values. The generated list replaces any existing lists in the user space.

Open Spooled File (QSPOPNSP) opens an existing spooled file. After the existing spooled file is opened, the Get Spooled File Data (QSPGETSP) API can then get the data and put it in the user space.

Put Spooled File Data (QSPPUTSP) puts data into a spooled file that was created using the Create Spooled File (QSPCRTSP) API. The data put in the spooled file is taken from a user space. The data in the user space can be created by either using the Get Spooled File Data (QSPGETSP) API or a user application.

Retrieve Spooled File Attributes (QUSRSPLA) returns specific information about a spooled file into a receiver variable. The size of the receiver variable determines the amount of information returned. You can specify the spooled file for which information is returned either with the internal job and spooled file identifiers, or with a specific job name, spooled file name, and spooled file number.

Print APIs

Print APIs can obtain information about or perform printing activities on the AS/400 system. Print APIs can:

- Retrieve output queue information such as status and number of entries on the queue.
- Retrieve information about specific printer writers.
- Transform data streams from one type to another.

The print APIs include the following:

Retrieve Output Queue Information (QSPROUTQ) retrieves information associated with the specified output queue. Information returned includes the parameters used to create the queue, the current status of the queue, and the number of entries on the queue.

Retrieve Writer Information (QSPRWTRI) retrieves printer writer information associated with the specified printer only when a printer writer is started to the printer. The information retrieved is similar to what can be seen when running the Work with Writer (WRKWTR) command for a particular printer writer.

Transform AFP to ASCII (QWPZTAFP) transforms an Advanced Function Printing data stream (AFPDS) into an ASCII data stream. This ASCII data stream can be formatted for IBM, Hewlett-Packard**, or PostScript**-capable printers.

Exit Program

Exit Program for Customized Separator Page allows a user to customize the separator page preceding a spooled file. For example, an exit program can be used to create a banner-style separator page.

Spool and Print Tools in Library QUSRTOOL

There are tools in QUSRTOOL that use some of the spooled file and print APIs. To show how to use these APIs, source code for these tools is provided in the examples.

These tools use most of the spooled file and print APIs discussed in this chapter. Additionally, some CL commands are used in the tools.

The tools are:

- Save and restore spooled files (SAVRSTSPLF) tool

  This tool uses the following spooled file APIs:

  - QSPCLOSP
  - QSPCRTSP
  - QSPGETSP
  - QUSLSPL
  - QSPOPNSP
  - QSPPUTSP

- Transform AFP to ASCII data stream (TRNAFP) tool

  This tool uses the QWPZTAFP print API.

- Customized separator page exit program (QSPBLSEP) tool

  This example of the customized print separator exit program builds program QSPBLSEP. QSPBLSEP can be used to generate data for separator pages.

A description of the SAVRSTSPLF tool can be found in member TSRINFO. A description of TRNAFP can be found in member TWPINFO. A description of QSPBLSEP can be found in member TBSINFO. All members are associated with file QATTINFO, library QUSRTOOL. File QATTINFO also contains information on how to use QUSRTOOL.

Close Spooled File (QSPCLOSP) API

Parameters

Required Parameter Group:

1	Spooled file handle returned	Input	Binary(4)
2	Error code	I/O	Char(*)

The Close Spooled File (QSPCLOSP) API closes a spooled file opened by the Open Spooled File (QSPOPNSP) API or created by the Create Spooled File (QSPCRTSP) API. A handle, which identifies the spooled file, is returned by the QSPOPNSP and QSPCRTSP APIs. This handle is used as source to the QSPCLOSP API.

A **spooled file handle** is an internal number that identifies a particular spooled file. The system assigns the handle when an existing spooled file is opened using the QSPOPNSP API or when a new spooled file is created using the QSPCRTSP API. The handle is valid until the spooled file is closed or the job ends.

Required Parameter Group

Spooled file handle returned
INPUT; BINARY(4)
The handle returned by the QSPOPNSP API or QSPCRTSP API.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF33DF E Internal data area for opened spooled files destroyed.
CPF33D2 E Spooled file handle not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Create Spooled File (QSPCRTSP) API

Parameters

Required Parameter Group:

1	Spooled file handle	Output	Binary(4)
2	Spooled file attributes	Input	Char(*)
3	Error code	I/O	Char(*)

The Create Spooled File (QSPCRTSP) API is used to create a new spooled file with attributes based on values taken from the spooled file attributes parameter. The field values of the spooled file attributes can be obtained using the Retrieve Spooled File Attributes (QUSRSPLA) API. The fields can also be defined by a user application.

Authorities and Locks

Special Authority
*SPLCTL. This authority is needed if you are creating a spooled file for another user.

Output Queue Authority
*USE

API QSPCRTSP Authority
PUBLIC(*EXCLUDE)

The API authority is set this way so that system administrators can control the use of this API. This API has security implications because you can create a spooled file from the data of another spooled file.

Required Parameter Group

Spooled file handle
OUTPUT; BINARY(4)
The handle to be used by subsequent put (QSPPUTSP API) and close (QSPCLOSP API) operations to identify the spooled file.

Spooled file attributes
INPUT; CHAR(*)
Attributes returned into the receiver variable by a prior Retrieve Spooled File Attributes (QUSRSPLA) API call with format SPLA0200 specified. These attributes are used to create the new spooled file. The current length of the attributes is 3301 bytes.

To see the SPLA0200 fields, go to "SPLA0200 Format Fields."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Considerations Using the QSPCRTSP API

The following special considerations apply when using the QSPCRTSP API.

System Values: When using the QSPCRTSP API, all system values must be uppercase for the AS/400 system to recognize them.

New Spooled File Ownership: The spooled file that is created by the QSPCRTSP API is spooled under one of two jobs. The job is determined by the user-name field. If the user name is the current user, it is a part of the user's job and is owned by the user profile that the job was started with.

However, ownership of the new spooled file can be assigned to a different user by specifying a different user profile name in the user-name field. The current user must have *SPLCTL authority to assign the spooled file to another user. When this is done, the new spooled file is by the user specified in the user-name field. The new spooled file is then part of a **special system job (QPRTJOB)** that is created for each user.

The user profile for the user name must already exist.

New Spooled File Placement: The new spooled file is placed on the output queue specified in the output queue name field. If the output queue is to correspond to the output queue of the user profile, the user application can use the Retrieve User Profile (RTVUSRPRF) command to retrieve the output queue name. The output queue name should be placed in the output queue fields before calling the QSPCRTSP API.

In both cases, the spooled file name is the one contained in the spooled file attributes parameter. The spooled file number is the next sequential one available for the job that the spooled file becomes a part of.

In both cases (if you are in the System/36 environment), the system assigns a new System/36 spooled file identifier and a new internal spooled file identifier.

Many of the fields returned by the Retrieve Spooled File Attributes (QUSRSPLA) API are closely interrelated with other fields. Changing these field values can lead to unexpected results.

Error Messages: Error messages can be CPIxxxx or CPFxxxx. Error messages that start with CPI are informational and do not stop the job from running. These messages are stored in the job log.

Error messages that start with CPF cause the job to stop running. These messages are returned to the application program. The form in which they are returned is determined by the error code parameter.

Error checking on the fields includes:

Note: If a default is used, that value is obtained from the device file.

- Length of fields is equal to the length of format SPLA0200.
- Special values for fields

  When a value is specified that is not valid, message CPI33E2 is issued and the field default is used.
- Incorrect values for fields

  When the field specifies a system name and the name is not valid, message CPF33E2 is issued. In the other cases, message CPI33E2 is issued and the default for that field is taken.
- Incorrect combinations of fields

  Message CPF33E2 is issued when the value of one field must correspond to a specific value of another field.

SPLA0200 Format Fields

The following table presents, in the same order, all of the fields of format SPLA0200. It contains the following entries:

- Field

  This is the field of the format.
- Used

  This column indicates whether this field is used in describing a new spooled file.

  Y This field is used in describing a new spooled file. Users creating their own spooled file data streams must set the Y fields.
  N This field is not used in describing a new spooled file.
- Cross-dependency

  This column indicates whether this field is dependent on other fields.

  Y This field should be changed with extreme caution. This field is closely interrelated with other fields and could cause unpredictable results if not set cor-

rectly. The new value will be reflected in the newly created spooled file.

N The field is easily changed, and this field is not related to or dependent on another field. The new value is reflected in the newly created spooled file.

N/A This field is not used; therefore, there is no need to change it.

- Description

  This column indicates relationships between fields and contains a brief description of how to change the field.

Figure 56-1 (Page 1 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Bytes returned	Y	N	The total number of bytes contained in the field format. This must be the length of format SPLA0200.
Bytes available	N	N/A	
Format name	Y	N	The name of the format of the fields. The only accepted value is SPLA0200.
Internal job identifier	N	N/A	
Internal spooled file identifier	N	N/A	
Job name	N	N/A	
User name	Y	Y	The user name of the owner of the new spooled file being created. If the user name is the current user, the spooled file is created as part of the current job. If the user name is someone other than the current user, the spooled file is created as part of the special system job QPRTJOB for the user name specified. The current user must have *SPLCTL authority.
Job number	N	N/A	
Spooled file name	Y	N	The name to be assigned to the new spooled file.
Spooled file number	N	N/A	
Form type	Y	N	The type of form loaded in the printer to print the spooled file.
User-specified data	Y	N	The 10 characters of user-specified data that describe the file.
Status	N	N/A	
File available	Y	N	The time this spooled file becomes available to an output device for processing. The possible values are: ***IMMED** The spooled file is available as soon as it is opened. ***FILEEND** The spooled file is available as soon as it is closed. ***JOBEND** The spooled file is available when the job that created it has completed. When *JOBEND is specified and the spooled file is being created for another user, the value is changed to *FILEEND because the job it is assigned to is the special system job QPRTJOB. When a value is not one of the three listed, the default used is *FILEEND. In this case the default is not taken from the device file specified.
Hold file before written	Y	N	Whether the spooled file is to be held. The possible values are: ***YES** The file is held. ***NO** The file is not held.
Save file after written	Y	N	Whether the spooled file is saved after it is written. The possible values are: ***YES** Save the spooled file. ***NO** Do not save the spooled file.
Total pages	Y	N	The number of pages this printer file contains.
Page or record being written	N	N/A	

Figure 56-1 (Page 2 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Starting page	Y	N	The page at which printing is to start for the spooled file. Special values include: **-1** Use the ending page value. **0** Printing starts on page 1.
Ending page	Y	N	The page at which printing is to end for the spooled file. 0 or 2147483647 indicates the last page. To print the entire spooled file, set this value to 0, 2147483647, or the last page number.
Last page printed	N	N/A	
Restart printing	Y	N	The number of the page where printing is restarted. The possible values are: **-1** *STRPAGE: The page specified in the starting page field. **-2** *ENDPAGE: The last page. **restart page** The page number where printing is restarted. This number should be within the page range specified by the spooled file starting and ending page field.
Total copies	N	N	Total copies to be produced for this spooled file.
Copies left to produce	Y	N	The number of copies remaining to be produced on the printer. Valid values are 1 through 255.
Lines per inch	Y	Y	The number (in tenths) of lines per vertical inch defined in the printer file. The value 40 indicates 4 lines per inch. Valid lines per inch include 4, 6, 8, 9, and 12. Lines per inch of 3 and 7.5 are valid for double-byte character set devices.
Characters per inch	Y	Y	The number (in tenths) of characters per horizontal inch, defined in the printer file. The value 100 indicates 10 characters per inch. Valid characters per inch include 5, 10, 12, 15, and 16.7. Characters per inch of 13.3, 18, and 20 are valid for double-byte character set devices.
Output priority	Y	N	The priority of the spooled file. The priority ranges from 1 (highest) to 9 (lowest). The priority specified must be within the limits of the owner of the spooled file. The limit can be found by using the Retrieve User Profile (RTVUSRPRF) command, specifying the PTYLMT parameter.
Output queue name	Y	N	The name of the output queue where the new spooled file is placed. If the output queue specified in the user profile for the owner of the spooled file is to be used, the Retrieve User Profile (RTVUSRPRF) command can be used, specifying the OUTQ parameter. A blank output queue name defaults to the printer file.
Output queue library name	Y	N	The name of the library that contains the output queue. The OUTQLIB parameter on the RTVUSRPRF can be used to retrieve the library name. A blank library name defaults to *LIBL.
Date file opened	N	N/A	
Time file opened	N	N/A	
Device file name	Y	Y	The name of the device file used to create the spooled file. The device file library name must also be specified.
Device file library name	Y	Y	The name of the library that contains the device file. The device file name must also be specified.
Program that opened file name	Y	Y	The name of the program that opened the spooled file. The library name of the program must also be specified.
Program library name that opened file	Y	Y	The name of the library that contains the program that opened the spooled file. The program name must also be specified.
Accounting code	Y	N	An identifier assigned by the system to record the resources used to write this spooled file. This can be retrieved for the user owning the spooled file by using the Retrieve User Profile (RTVUSRPRF) command, specifying the ACGCDE parameter.
Print text	Y	N	The text that is printed at the bottom of each page and on separator pages.
Record length	Y	Y	The length of spooled file records. If the field shows -1 (the special value *RCDFMT), this is an externally defined spooled file, and the length is included in the spooled file definition. The length includes the extra length of 1 for carriage control if it exists.

Figure 56-1 (Page 3 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Maximum records	Y	N	The maximum number of records allowed in the spooled file when it is opened. Valid values are 1 through 500000. A special value of 0 indicates *NOMAX.
Device type	N	N	
Printer device type	Y	Y	The type of data stream used to represent the spooled file. The possible values are: ***AFPDS** Advanced Function Printing* data stream ***AFPDSLINE** AFPDS data mixed with 1403 line data ***IPDS** Intelligent printer data stream ***LINE** 1403 line data ***SCS** Systems Network Architecture (SNA) character stream ***USERACSII** ASCII data The data stream of the spooled file must correspond to the printer device type.
Document name	Y	Y	The name of the document that was the source of the spooled file. This field is used by the OfficeVision/400 program and must be blank for other spooled files. If the document name is specified, the folder name must also be specified.
Folder name	Y	Y	The name of the folder that contains the document name. This field is used by the OfficeVision/400 program and must be blank for other spooled files. If the folder name is specified, the document name must also be specified.
System/36 procedure name	Y	N	The name of the procedure running when the spooled file is created.
Print fidelity	Y	Y	The kind of error handling that is performed when printing spooled files with a printer device type of *AFPDS. The possible values are: ***CONTENT** The printing overrides errors in the data stream and continues printing with the printers best quality based on the content fidelity. ***ABSOLUTE** The spooled file is printed exactly as intended. Printing is stopped if an error is encountered in the data stream. This field should be set to *CONTENT for all non-AFPDS files.
Replace unprintable characters	Y	N	Whether characters that cannot be printed are to be replaced with another character. Valid values are Y (yes) or N (no).
Replacement character	Y	Y	The character that replaces any unprintable characters. This field has a value if the replace unprintable characters field specifies Y.
Page length	Y	Y	The page length (in lines per page) used by the spooled file specified in the device file field. The valid range is row 1 through 255. The value should not exceed the actual length of the page.
Page width	Y	Y	The page width (in characters per printed line) used by the spooled file specified in the device file field. The valid range is column 1 through 378, although some printers have a page width less than 378. The value should not exceed the actual width of the page used.
Number of separators	Y	N	The number of file separator pages placed at the beginning of each copy. Valid values are 0 through 9, with 0 meaning no separator pages.
Overflow line number	Y	Y	The last line to be printed before the data being printed overflows to the next page. Valid values are 1 through 255. This value cannot be greater than the value specified in the page length field.
Double-byte character set (DBCS) data	Y	Y	Whether this spooled file contains double-byte character set (DBCS) data. The options are *YES or *NO. The other DBCS fields should also be set appropriately.
DBCS extension characters	Y	Y	Whether the system is to use the extension character processing function. The possible values are: ***YES** The system processes DBCS extension characters. ***NO** The system does not process DBCS extension characters; it prints extension characters as the undefined character. This field should be *NO when the DBCS data field is *NO.

Figure 56-1 (Page 4 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
DBCS shift-out shift-in (SO/SI) spacing	Y	Y	The presentation of shift-out and shift-in characters when printed. The possible values are: ***YES** Shift-out and shift-in characters occupy one space. ***NO** Shift-out and shift-in characters occupy no space. ***RIGHT** Shift-out characters occupy no space and shift-in characters occupy two spaces. This field should be *NO when the DBCS data field is *NO.
DBCS character rotation	Y	Y	Whether this printer file causes the DBCS characters to be rotated 90 degrees counterclockwise before printing. The possible values are *YES and *NO. This field should be *NO when the DBCS field is *NO.
DBCS characters per inch	Y	Y	The number of double-byte characters to be printed per inch. The possible values are: **-1** *CPI: One-half of the characters-per-inch value. **-2** *CONDENSED: 20 double-byte characters per 3 inches. **5** 5 characters per inch. **6** 6 characters per inch. **10** 10 characters per inch. This field should be -1 when the double-byte character set field is *NO.
Graphic character set	Y	Y	The set of graphic characters used when printing the spooled file. For *DEVD, the system gets the graphic character set from the printer device description. Valid values are 1 through 32767 and *DEVD. When a numeric value is specified, it is represented as a character string that is right-justified with leading zeros.
Code page	Y	Y	The mapping of graphic characters to code points for this printer. For *DEVD, the system gets the code page from the printer device description and this value is ignored. Valid values are 1 through 32767 and *DEVD. When a numeric value is specified, it is represented as a character string that is right justified with leading zeros.
Form definition name	Y	Y	The name of the form definition to use for printing the spooled file. The special value *DEVD is the default. See note 3 on page 56-19 for additional information about this field.
Form definition library name	Y	Y	The name of the library that contains the form definition. The special value *LIBL should be specified when the form definition name field is specified as *DEVD. See note 3 on page 56-19 for additional information about this field.
Source drawer	Y	Y	The drawer to be used when the form feed field is set to *AUTOCUT. The possible values are: **1-255** The drawer number. **-1** *E1, the envelope drawer. **-2** *FORMDF, indicating that the file uses a user-specified form definition. This value is used only when the printer device type (DEVTYPE) field has been set to *LINE, *AFPDS, or *AFPDSLINE.
Printer font	Y	Y	The printer font that is used. The possible values are: ***DEVD** The font defined in the printer device description. ***CPI** A font that has the pitch specified by the characters-per-inch field. **identifier** The numeric font identifier that is used with the printer device file. The font specified should be valid for the printer.
System/36 spooled file identifier	N	N/A	

Figure 56-1 (Page 5 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Page rotation	Y	N	The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. The possible values are: **-1** *AUTO: Computer output reduction is done automatically if the output is too large to fit on the form, regardless of the print quality. **-2** *DEVD: Page rotation is obtained from the printer device description. **-3** *COR: Output created for a form 13.2 inches wide by 11.0 inches long is adjusted to print on a form 11.0 inches wide by 8.5 inches long. **0** No rotation is done. Printing starts at the edge of the paper loaded into the printer first and is parallel to that edge. **90** Text is rotated 90 degrees clockwise from the 0-degree writing position. **180** Text is rotated 180 degrees clockwise from the 0-degree writing position. **270** Text is rotated 270 degrees clockwise from the 0-degree writing position.
Justification	Y	N	The percentage that the output is right-justified. The values are 100, 50, or 0.
Print on both sides (duplex)	Y	Y	How the information prints. The possible values are: ***FORMDF** A user-specified form definition determines how printing is done. This value is used only when the printer device type (on the printer file) is specified as *LINE, *AFPDS, or *AFPDSLINE. ***NO** The print appears on only one side of a piece of paper. ***YES** The print is on both sides of the page with the top of each page the same for both sides. ***TUMBLE** The print is on both sides of the page with the top of one printed page at the opposite end from the top of the other printed page.
Fold records	Y	N	Whether all positions in a record are displayed. The possible values are: ***YES** Records whose length exceeds the page width flow to the following line. ***NO** Records whose length exceeds the page width are truncated.
Control character	Y	Y	Whether the data to be printed used the American National Standard printer control character. The possible values are: ***NONE** No print control characters are passed in the data that is printed. ***FCFC** The first character of every record is an American National Standard printer control character. The record length field must include one position for forms control code. This value is not valid for externally described printer files.
Align forms	Y	N	Whether a forms alignment message is sent prior to printing the data. The options are *YES or *NO.
Print quality	Y	N	The print quality used when printing the data. The possible values are: ***STD** Standard ***DRAFT** Draft ***NLQ** Near-letter quality ***FASTDRAFT** Pages printed per minute are greater than if *DRAFT is specified.
Form feed	Y	N	The form feed attachment used by the printer device file. The possible values are: ***DEVD** The forms are fed into the printer as described in the device description. ***CONT** Continuous forms are used by the printer. The tractor-feed attachment must be put on the printer if this value is specified. ***CUT** Single-cut sheets are used by the printer. Each sheet is manually loaded. The forms alignment message is not sent for cut sheets. ***AUTOCUT** Single-cut sheets are semiautomatically fed into the printer. The sheet-feed attachment must be put on the printer if this value is specified. The forms alignment message is not sent for cut sheets.
Volumes (array)	N	N/A	Diskette information is not valid for these APIs.
File label identifier	N	N/A	Diskette information is not valid for these APIs.
Exchange type	N	N/A	Diskette information is not valid for these APIs.
Character code	N	N/A	Diskette information is not valid for these APIs.

Figure 56-1 (Page 6 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Total records	Y	N	The number of data records. This field is only used when the device type (DEVTYPE) parameter of the printer file is *AFPDS, *LINE, or *AFPDSLINE.
Multiple up (pages per side)	Y	N	The number of logical pages that print on each side of each physical page when this spooled file prints. The possible values are 1, 2, and 4. See note 1 on page 56-19 for additional information about this field.
Front overlay name	Y	Y	The name of the **front overlay** (the material that prints on the front side of each page). The possible values are: ***NONE** The file does not use the front overlay. **front overlay name** The name of the front overlay. If an overlay name is specified, the front overlay library name must also be specified. See note 1 on page 56-19 for additional information about this field.
Front overlay library name	Y	Y	The name of the library containing the front overlay. The possible values are: ***CURLIB** The current library for the job locates the front overlay. ***LIBL** The library list locates the front overlay. **library name** This library is searched for the front overlay. See note 1 on page 56-19 for additional information about this field.
Front overlay offset down	Y	Y	The offset down from the point of origin where the overlay is printed. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 1 on page 56-19 for additional information about this field.
Front overlay offset across	Y	Y	The offset across from the point of origin where the overlay is printed. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 1 on page 56-19 for additional information about this field.
Back overlay name	Y	Y	The name of the **back overlay** (the material that prints on the back side of each page). The possible values are: ***FRONTOVL** The back overlay is the same as the front overlay. ***NONE** The file does not use a back overlay. **back overlay name** The name of the back overlay. If an overlay name is specified, the back overlay library name must also be specified. See note 1 on page 56-19 for additional information about this field.
Back overlay library name	Y	Y	The name of the library containing the back overlay. The possible values are: ***CURLIB** The current library for the job locates the back overlay. ***LIBL** The library list locates the back overlay. **library name** This library is searched for the back overlay. See note 1 on page 56-19 for additional information about this field.
Back overlay offset down	Y	Y	The offset down from the point of origin where the overlay is printed. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 1 on page 56-19 for additional information about this field.
Back overlay offset across	Y	Y	The offset across from the point of origin where the overlay is printed. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 1 on page 56-19 for additional information about this field.

Figure 56-1 (Page 7 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Unit of measure	Y	Y	The unit of measure to use for specifying distances. The unit of measure is used with certain parameters of the printer file. Page definitions and form definitions are examples of these parameters. The possible values are: ***CM** Centimeters ***INCH** Inches
Page definition name	Y	Y	The name of the page definition to use for the spooled file. This information is used only when the printer device type (DEVTYPE) parameter is *LINE or *AFPDSLINE. The special value of *NONE indicates no page definition. When a page definition name is specified, the page definition library name must also be specified.
Page definition library name	Y	Y	The name of the library in which the page definition resides. This information is necessary only when the printer device type (DEVTYPE) parameter is *LINE or *AFPDSLINE.
Line spacing	Y	Y	How a spooled file's line data records are spaced when printed. This information is necessary only when the printer device type (DEVTYPE) parameter on the printer file is *LINE or *AFPDSLINE. The possible values are: ***SINGLE** The printed output is single spaced. If the data contains control characters, they are ignored. ***DOUBLE** The printed output is double spaced. If the data contains control characters, they are ignored. ***TRIPLE** The printed output is triple spaced. If the data contains control characters, they are ignored. ***CTLCHAR** The control characters in the data determine the line spacing. This value should be used when the printer device type is not *LINE or *AFPDSLINE.
Point size	Y	Y	The point size in which this spooled file's characters should be printed. Valid values are 000.1 through 999.9. This field is ignored if the print font is specified as 0.0 (*DEVD).
Maximum spooled data record size	Y	Y	The length of the largest record in the spooled file. For DEVTYPE *LINE, *AFPDSLINE, and *AFPDS spooled files, this length is the length of the largest record in the spooled file. For all other printer device types (DEVTYPE), this length is the same as the spooled file buffer size.
Spooled file buffer size	Y	N	The length of the buffer being written. Sizes supported are 512 and 4079. A buffer size of 512 is recommended for spooled files that have a schedule field of *IMMED. A buffer size of 4079 is recommended for spooled files that have a schedule field of *FILEEND or *JOBEND. If the original spooled file was spooled with a buffer size of 4079, changing the buffer size to 512 causes a function check to occur.
Spooled file level	Y	N	The level of the spooled file in Version, Release, and Modification level format (VxRxMx).
Coded font array	Y	Y	The name of the fonts used when printing a spooled file with a printer device type (DEVTYPE) of *LINE or *AFPDSLINE. The name does not include the 2-character prefix of the coded font name (X0-XG). This array should be blank for all other printer device types.
Channel mode	Y	Y	A variable indicating the channel value mode. They are: ***NORMAL** Use the normal channel values (1 equals skip to line 1, 2 through B equal space one line before printing, C equals skip to overflow line). **blank** The channel values specified in the channel value array are used.
Channel value array	Y	Y	The array contains 12 channels. Each channel is assigned a number, 1 through 12. Each channel (or number in the array) is assigned a value. That value is the line that the printer skips to before printing starts. For example, if channel 9 had a value of 9, the printer skips to line 9 before printing starts. When creating a file with data previously spooled and channel values specified, changing the channel values does not affect the newly created spooled file. The channel values used when printing are the ones specified when the data was originally spooled.
Graphics token	Y	Y	The printer type on which the graphics in the spooled file can be printed. The possible values are: **4214** Data stream contains graphics for a 4214 printer. **4234** Data stream contains graphics for a 4234 printer. **522X** Data stream contains graphics for a 522x printer. **IPDS** Data stream contains graphics for an IPDS printer.

Figure 56-1 (Page 8 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Record format	Y	Y	The format of the records in the spooled file. The possible values are: ***FIXED** All records in the spooled file are the same size; that is, the original length specified in the print data for each record is the same for all records. ***VARIABLE** Records in the spooled file vary in size; that is, the original length specified in the print data for each record has at least one different value for records in the spooled file. For DEVTYPE *USERASCII, *IPDS, and *SCS without ASCII data files are all fixed format. *SCS files with ASCII data are variable length. For DEVTYPE *AFPDS, *AFPDSLINE, and *LINE, data files can be either fixed or variable.
Height of drawer 1	Y	Y	The height in inches of the paper in drawer 1. This field is set to packed 0 when the spooled file is being created as opposed to being copied. All other drawer fields should also be set to packed 0.
Width of drawer 1	Y	Y	The width in inches of the paper in drawer 1. This field is set to packed 0 when the spooled file is being created as opposed to being copied. All other drawer fields should also be set to packed 0.
Height of drawer 2	Y	Y	The height in inches of the paper in drawer 2. This field is set to packed 0 when the spooled file is being created as opposed to being copied. All other drawer fields should also be set to packed 0.
Width of drawer 2	Y	Y	The width in inches of the paper in drawer 2. This field is set to packed 0 when the spooled file is being created as opposed to being copied. All other drawer fields should also be set to packed 0.
Number of buffers	N	N/A	The number of buffers in the spooled file.
Maximum forms width	Y	N	The maximum width of printer file forms in character positions.
Alternate forms width	Y	Y	The width of the alternate forms in character positions. This field applies only to spooled files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields. The other alternate forms fields should also be set to 0.
Alternate forms length	Y	Y	The length of the alternate forms in lines. This field applies only to spooled files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields. The other alternate forms fields should also be set to 0.
Alternate lines per inch	Y	Y	The lines per inch for the alternate forms. This field applies only to spooled files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields. The other alternate forms fields should also be set to 0.
System/38 Text Utility flags	Y	N	The flag for advanced functions. This field is set to hexadecimal zeros when a spooled file is being created as opposed to copying an existing file.
File open	Y	N	Whether the spooled file is still open when the fields are retrieved. If fields retrieved from an open spooled file are used to create a new spooled file, the create operation fails. Changing this flag to N for a spooled file that is open could cause unpredictable results in a newly created spooled file.
Page count estimated	Y	Y	Whether the total pages given is an estimate. Valid values are Y (yes) or N (no). The total page count is estimated in the following situations: • If the spooled file device type is *AFPDSLINE. • If the spooled file device type is *LINE and a page definition was used. • If the spooled file device type is *AFPDS and the spooled file was not created on an AS/400 system. • If the spooled file device type is *USERASCII.
File stopped printing on page boundary	Y	N	Whether the spooled file stops on a page boundary. Valid values are Y (yes) or N (no). If the spooled file has not been sent to a printer, set the field to N.

Figure 56-1 (Page 9 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
TRC for 1403	Y	Y	Whether the file contains 1403 line data with table-reference characters (TRC). This field is only used for device types (DEVTYPE) *LINE and *AFPDSLINE. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Define characters	Y	Y	Whether the spooled file defines or redefines unique print characters. For DEVTYPE(*SCS), a Load Alternate Characters command is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Characters per inch changes	Y	Y	Whether the spooled file changes the characters per inch (cpi) within the file. For DEVTYPE(*SCS), a Set Character Distance command is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Transparency	Y	Y	Whether the SCS Transparency command is used in the spooled file. It is set to N for all other device type files, such as IPDS. For DEVTYPE(*SCS), a TRANSPARENT SCS command is contained in the spooled file. Valid values are Y (yes) or N (no).
Double-wide character	Y	Y	Whether the spooled file prints everything twice as wide as normal. For DEVTYPE(*SCS), a Set Font Size command is contained in the data stream. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
DBCS character rotation command	Y	Y	Whether the double-byte character set characters are rotated 90 degrees counterclockwise before printing. • For DEVTYPE(*SCS), a Set Text Orientation command is contained in the spooled file. • For DEVTYPE(*AFPDS), a map coded font-1 (MCF-1) structured field that has the character rotation parameter set to hex 8700 is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Extended code page	Y	Y	Whether the extended code page changes within the file. • For DEVTYPE(*SCS), a Set CGCS through GCID is contained in the spooled file. • For DEVTYPE(*IPDS), a Load Font Equivalence command is inserted into the Load Font Equivalence table. • For DEVTYPE(*AFPDS), a map code font-2 (MCF-2) structured field is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
FFT emphasis	Y	Y	Whether the spooled file contains text that is to appear darker than the surrounding text. For DEVTYPE(*SCS), Begin Emphasis and End Emphasis commands are contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
3812 SCS	Y	Y	Whether the spooled file contains fonts supported only on the 3812 printer. For DEVTYPE(*SCS), a Set Coded Font Local command with a global ID of greater than 255 is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Set line density command	Y	Y	Whether the lines per inch (lpi) changes within the spooled file or is specified with the Set Line Density SCS command. For DEVTYPE(*SCS), a Set Line Density command is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Graphics error actions	Y	Y	Whether the file contains graphic error action commands. For SCS files, the file contains one or more Set Graphic Error Action commands. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.

Figure 56-1 (Page 10 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
5219 commands	Y	Y	Whether the file contains commands that are not valid on a 5219 printer. The data stream contains one or more of the following SCS commands, which either are not valid on a 5219 or contain parameters that are not valid. • An incorrect Set Character Set Global ID command is found by validity checking the code page with what is valid for the 5219 printer. • A Begin and End Emphasis SCS commands. • A Set Presentation Page Size command contains a page size value that is not supported. • A Page Presentation Media command contains a drawer value other than drawer 1 or drawer 2. • A Set Single Line Distance command contains a distance value that is not supported. • A Set Single Line Spacing command contains a distance value that is not supported. • A Justify Text Field Format command contains a value other than 0, 50, or 100 percent. • A Set Justify Mode Format command contains a value other than 0, 50, or 100 percent. • A Set Presentation Color command. • A Begin Overstrike command that contains an optional CHRID value other than 0. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
3812 SCS commands	Y	Y	Whether the file contains commands that are not valid on the 3812 printer. • An incorrect Set Character Set Global ID command is found by validity checking the code page with what is valid for the 3812 printer. • An incorrect Set Character Set Local ID command is found by validity checking the code page with what is valid for the 3812 printer. • A Set Presentation Color command Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Field outlining	Y	Y	Whether the spooled file outlines fields of data with boxes. If it does, the file must be printed on a printer that supports field outlining. For DEVTYPE(*SCS), a Define Grid Line command is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Final form text	Y	Y	Whether this spooled file contains various functions that are supported on letter-quality printers. • For DEVTYPE(*SCS), a Document Content Architecture (DCA) command is contained in the spooled file. DCA commands include: – Required New Line – Required Form Feed – Indent Tab – Set Presentation Page Size – Set Horizontal Margins – Set Vertical Margins – Release Left Margin – Set Line Spacing – Set Single Line Distance – Justify Text Field Format – Set Justify Mode – Set Horizontal Tab Stops – Set Indent Level – Set Exception Action – Set Presentation Color – Set Spacing Variable – Begin and End Overstrike – Begin and End Underscore – Bell – Switch – Repeat – Tab – Backspace – Unit Backspace – Substitute – Word Underscore Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.

Figure 56-1 (Page 11 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Bar code	Y	Y	Whether the spooled file contains bar codes created using the BARCODE keyword in the data description specifications (DDS). For DEVTYPE(*IPDS and *AFPDS), a series of bar code commands is contained in the data stream. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Color	Y	Y	Whether an IPDS Write Text command or AFPDS presentation text data (PTX) structured field containing a Set Text Color control is contained in the spooled file. For DEVTYPE(*IPDS and *AFPDS), a Set Text Color text control is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Drawer change	Y	Y	Whether the printer drawer is changed within the spooled file. • For DEVTYPE(*SCS), a Page Presentation Media command is contained in the spooled file. • For DEVTYPE(*IPDS), an Execute Order Home State-Select Input Media Source command is in the spooled file. • For DEVTYPE(*AFPDS), a media map and an invoke media map are in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Character ID	Y	Y	Whether the character ID can change within the file. • For DEVTYPE(*SCS), a Set CGCS through Local ID command is contained in the spooled file. • For DEVTYPE(*IPDS), a Load Font Equivalence command is inserted into the Load Font Equivalence table. • For DEVTYPE(*AFPDS), a map code font-2 (MCF-2) structured field is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Lines per inch changes	Y	Y	Whether the lines per inch (lpi) changes within the spooled file. • For DEVTYPE(*SCS), a Set Single Line Density command is contained in the spooled file. • For DEVTYPE(*IPDS), a Load Page Descriptor command is contained in the spooled file, which specifies the line-per-inch value • For DEVTYPE(*AFPDS), a presentation text description (PTD) structured field that specifies the baseline increment is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Font	Y	Y	Whether the spooled file uses multiple fonts. • For DEVTYPE(*SCS), a Set Font Global command is in the spooled file. • For DEVTYPE(*IPDS), a Load Font Equivalence command is contained in the Load Font Equivalence table. • For DEVTYPE(*AFPDS), a map coded font-1 (MCF-1) or map coded font-2 (MCF-2) structured field is in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Highlight	Y	Y	Whether the spooled file contains text that is to appear darker than the surrounding text. • For DEVTYPE(*SCS), data is overprinted to get the bold effect. • For DEVTYPE(*IPDS), a Load Font Equivalence command is contained in the Load Font Equivalence table, which has the bold field set on. • For DEVTYPE(*AFPDS), a map coded font-2 (MCF-2) structured field that has the font weight class parameter set to hex 07 is in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.

Figure 56-1 (Page 12 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Page rotate	Y	Y	Whether the spooled file changes the page rotation to be used within the file. • For DEVTYPE(*SCS), a Set Text Orientation command is contained in the spooled file. • For DEVTYPE(*IPDS), a Load Page Descriptor command is contained in the spooled file. See the *Intelligent Printer Data Stream Reference* for detailed information about rotating text in a spooled file. • For DEVTYPE(*AFPDS), a presentation text descriptor (PTD) structured field that specifies the page rotation is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Subscript	Y	Y	Whether the spooled file contains subscript. • For DEVTYPE(*SCS), a Subscript command is contained in the spooled file. • For DEVTYPE(*IPDS and *AFPDS), a Temporary Baseline Move text control is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Superscript	Y	Y	Whether the spooled file contains superscript. • For DEVTYPE(*SCS), a Superscript command is contained in the spooled file. • For DEVTYPE(*IPDS and *AFPDS), a Temporary Baseline Move text control is contained in the spooled file. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
DDS	Y	Y	Whether the file was constructed with DDS. Valid values are Y (yes) or N (no). This field should be set to N when the file contains data that has not been spooled before.
Final form feed	Y	Y	Whether the Final Form Feed command is in the printer file. Valid values are Y (yes) or N (no).
SCS data	Y	Y	Whether the spooled file is created with SCS already in the input data. Valid values are Y (yes) or N (no).
User-generated data stream	Y	Y	Whether the data stream has been validated by a system program on the AS/400 system when the data was spooled, for example, the OfficeVision/400 program. Valid values are Y (yes) or N (no). This field should be set to Y (data stream not validated) when opening a spooled file that contains data that has not been spooled, for example, OfficeVision/400 spooled files.
Graphics	Y	Y	Whether the spooled file contains graphics commands in its data stream. Valid values are Y (yes) or N (no).
Unrecognizable data	Y	Y	Whether the file contains SCS commands that are not valid because of one of the following: • Command is not recognized • Command data is not valid • Command is recognized, but not supported Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
ASCII transparency	Y	Y	For SCS files, whether ASCII commands are embedded in the ASCII transparency command. The ASCII transparency command is command_ID/command_length/ASCII_data. The command ID is a 1-byte field with the value hex 03. The command length is a 1-byte field that contains the length of the command length field and the ASCII data field. Valid values are Y (yes) or N (no). This field should be set to N when creating a spooled file as opposed to copying an existing spooled file.
IPDS transparent data	Y	N	Whether the file contains data from System/36 Programming Request for Price Quotations (PRPQs). Valid values are Y (yes) or N (no). This field should be set to N when creating a spooled file as opposed to copying an existing spooled file.
OfficeVision/400	Y	N	Whether the spooled file is generated by the OfficeVision/400 licensed program. Valid values are Y (yes) or N (no). This field should be set to N when creating a spooled file as opposed to copying an existing spooled file.

Figure 56-1 (Page 13 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Lines-per-inch (lpi) value not supported	Y	N	This field applies only to OfficeVision/400 files. Valid values are Y (yes) or N (no). It will be N for all other spooled files. The lines-per-inch (lpi) value is designed to support 1440 lines per inch. To be able to visually read any data, you would have to select a value that creates a fraction that represents the number of lines printed per inch. For example, if the lpi value were 100, you would get 14.4 lines per inch (1440 divided by 100).
CPA3353 message	Y	N	Whether message CPA3353 is issued when this spooled file is printed. Valid values are Y (yes) or N (no).
Set exception	Y	Y	Whether the spooled file has the SCS Set Exception Action command in the data stream. Valid values are Y (yes) or N (no).
Carriage control character	Y	Y	Whether the data contains carriage control characters. This field is valid only when the printer device type (DEVTYPE) field is *LINE or *AFPDSLINE. Valid values are Y (yes) or N (no). Set this field to N for all other printer device types.
Page position	Y	Y	Whether page positioning errors are reported. This field is valid only when the printer device type (DEVTYPE) field is *LINE, *AFPDSLINE, or *AFPDS. Valid values are Y (yes) or N (no). This field is set to N for all other printer device types.
Character not valid	Y	Y	Whether incorrect character errors are reported. This field is valid only when the printer device type (DEVTYPE) field is *LINE, *AFPDSLINE, or *AFPDS. Valid values are Y (yes) or N (no). This field is set to N for all other printer device types.
Lengths present	Y	Y	Whether the 8-byte length information is present in the print text data buffers for format SPLF0200. This field is valid when the printer device type (DEVTYPE) field is *AFPDS, *AFPDSLINE, or *LINE. Valid values are Y (yes) or N (no). It must be set to Y for *AFPDSLINE and *LINE printer device types. *AFPDS data may or may not have the length information. This field should be set to N for all other printer device types.
5A present	Y	Y	Whether the hex 5A constant is present in all AFPDS data for the file (in format SPLF0200). This field is valid when the printer device type (DEVTYPE) field is *AFPDS. Valid values are Y (yes) or N (no). This field should be set to N for all other printer device types.
Number of font array entries	Y	Y	The number of font equivalence array entries used. The valid values for the field are 0 through 48, where 0 indicates that there are no entries in the array. The field is valid only for spooled files that have the printer device type field equal to *IPDS. It should be set to zero for all other printer device types.
Number of resource library entries	N	N	The number of entries in the resource library array used. The valid values for the field are 0 through 43, where 0 indicates that there are no entries in the array.
Font equivalence array	Y	Y	The data portion of the Load Font Equivalence (LFE) command for intelligent printer data streams (IPDS). The current maximum is 48. However, for future expansion, the array returns 72 sixteen-character font equivalences. To support that further expansion, the 1153rd character of the variable is set to 1 when more than 72 entries are available. The format of this entry is described in the *Intelligent Printer Data Stream Reference*.
Resource library array	N	N	The creation of this spooled file is probably based on attributes retrieved from an existing spooled file. The library names in the resource library array are the names of libraries used when the spooled file was originally created. They have no effect on the creation of a new spooled file when using the QSPCRTSP API. If you need resources to be used with the new spooled file, they must be in libraries contained in your library list at spooled file creation time.
AS/400-created AFPDS	Y	Y	Whether the spooled file contains AFPDS data created on the AS/400 system. Valid values are Y (yes) or N (no).
Job character ID specified	Y	Y	Whether the graphic character set and code page of the spooled file is taken from the coded character set identifier (CCSID) of the job. The possible values are Y or N.

Figure 56-1 (Page 14 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Front margin offset down	Y	Y	For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are: **-2** *DEVD. For printers configured AFP(*YES), no print border is used for front margin offsets across and down. Otherwise, the offset values are 0. **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 2 on page 56-19 for additional information about this field.
Front margin offset across	Y	Y	For the front side of a piece of paper it, specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 2 on page 56-19 for additional information about this field.
Back margin offset down	Y	Y	For the back side of a piece of paper, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are: **-1** The back margin offsets are the same as the front margin offsets. **-2** *DEVD. For printers configured AFP(*YES), no print border is used for back margin offsets across and down. Otherwise, the offset values are 0. **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 2 on page 56-19 for additional information about this field.
Back margin offset across	Y	Y	For the back side of a piece of paper, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far from the left side of the page printing starts. The possible values are: **0–57.79** The offset in inches or centimeters, according to the unit of measure field. The maximum offset is 57.79 centimeters or 22.75 inches. See note 2 on page 56-19 for additional information about this field.
Length of page	Y	Y	The length of a page in rows and columns (*ROWCOL) or in inches and centimeters (*UOM). If this field is specified and the measurement method used is *ROWCOL, this value should be the same as that specified on the page length field found in this format. If this field is specified and the measurement method is *UOM, the actual distance specified must be equivalent to that specified on the page length field found in this format. If the measurement method is *ROWCOL, the page length value must be from 1 through 255. If the measurement method is *UOM, the page length value must be between 0 and 57.79.
Width of page	Y	Y	The width of a page in rows and columns (*ROWCOL) or in inches and centimeters (*UOM). The width of a page is determined by the value specified on the PAGESIZE parameter of the printer file. If this field is specified and the measurement method used is *ROWCOL, this value should be the same as that specified on the page width field found in this format. If this field is specified and the measurement method is *UOM, this value must be equivalent to that specified on the page width field found in this format. If the measurement method is *ROWCOL, the page width value must be from 1 through 378. If the measurement method is *UOM, the page width value must be between 0 and 57.79.
Measurement method	Y	Y	The measurement method used for the length of page and width of page fields. The possible values are: ***ROWCOL** Uses rows and columns as the units of measure. ***UOM** Uses the value specified on the unit of measurement parameter (UOM). UOM is either inches (*INCH) or centimeters (*CM).

Figure 56-1 (Page 15 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
Advanced Function Printing (AFP*) resource	Y	Y	Whether this spooled file refers to AFP resources (using DDS) external to this spooled file, for example, page segments or overlays. The possible values are: **Y** The spooled file refers to AFP resources external to the spooled file. **N** The spooled file does not refer to AFP resources external to the spooled file.
Character set name	Y	Y	The name of the font character set object used to print this file. The possible values are: ***FONT** The information specified on the font parameter is used instead of the character set and code page. **character set name** The name of the font character set object to use. See note 4 on page 56-19 for additional information about this field.
Character set library name	Y	Y	The name of the library containing the font character set object. The possible values are: ***LIBL** The library list is used to locate the font character set object. **library name** This library is searched for the font character set object. See note 4 on page 56-19 for additional information about this field.
Code page name	Y	Y	The name of the code page used to print this spooled file. See note 4 on page 56-19 for additional information about this field.
Code page library name	Y	Y	The name of the library containing the code page used to print this spooled file. The possible values are: ***LIBL** The library list is used to locate the code page. **library name** This library is searched for the code page. See note 4 on page 56-19 for additional information about this field.
Coded font name	Y	Y	The name of the coded font used to print this spooled file. A **coded font** is an AFP resource composed of a font character set and a code page. The possible values are: ***FNTCHRSET** The values used are the values specified on the character set and code page fields. **coded font name** The name of the coded font used to print this spooled file. See note 4 on page 56-19 for additional information about this field.
Coded font library name	Y	Y	The name of the library containing the coded font used to print this spooled file. The possible values are: ***LIBL** The library list is used to locate the coded font. **library name** This library is searched for the coded font. See note 4 on page 56-19 for additional information about this field.
DBCS-coded font name	Y	Y	Specifies the name of the DBCS-coded font used to print DBCS data on printers configured as AFP(*YES). The possible values are: ***SYSVAL** The DBCS-coded font specified in the system value is used. **DBCS-coded font name** The name of the DBCS-coded font being used. See note 5 on page 56-19 for additional information about this field.
DBCS-coded font library	Y	Y	The name of the library containing the DBCS coded font used to print this spooled file. The possible values are: ***CURLIB** The current library is searched for the DBCS-coded font. ***LIBL** The library list is used to locate the DBCS-coded font. **Library name** This library is searched for the DBCS-coded font. See note 5 on page 56-19 for additional information about this field.

Figure 56-1 (Page 16 of 16). SPLA0200 Format Field Names and Descriptions

Field	Used	Cross-Dependency	Description
User-defined file	N	N	Whether the spooled file was created by using an API. The possible values are: ***YES** The spooled file was created using the QSPCRTSP API. ***NO** The spooled file was created by normal system functions.

Notes:

1 This field should only be specified for spooled files with device types of *SCS, *IPDS, or *AFPDS if the spooled file was created on an AS/400 system. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

2 This field should only be specified for spooled files with device types of *LINE, *AFPDSLINE, or *AFPDS if the spooled file was created on an AS/400 system. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

3 This field should only be specified for spooled files with device types of *LINE, *AFPDSLINE, or *AFPDS if the spooled file was not created on an AS/400 system. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

4 This field should only be specified for spooled files that contain *AFPDS created on the AS/400 system. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

5 This field should only be specified on a DBCS system, and only for spooled files that will be created with a device type of *SCS or that will contain AS/400-created *AFPDS. See the AS/400-created *AFPDS field.

If this field is not specified for one of these valid types, a default value is used.

Error Messages

| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3C19 E Error occurred with receiver variable specified.
| CPF3C21 E Format name &1 is not valid.
| CPF33DD E Maximum number of open spooled files exceeded for this job.
| CPF33DE E Size of internal data for opened spooled file exceeds maximum.
| CPF33DF E Internal data area for opened spooled files destroyed.
| CPF33E0 E Incomplete set of attributes provided.
| CPF33E1 E Attributes are for an opened spooled file.
| CPF33E2 E Value &1 for spooled file attribute at offset &2 not valid.
| CPF34B1 E Spooled file print data format not supported.
| CPF34B2 E Spooled file input record length is not valid.
| CPF9838 E User profile storage limit exceeded.
| CPF9845 E Error occurred while opening file &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Get Spooled File Data (QSPGETSP) API

Parameters

Required Parameter Group:

1	Spooled file handle	Input	Binary(4)
2	User space name and library	Input	Char(20)
3	Format name	Input	Char(8)
4	Ordinal number of buffer to be read	Input	Binary(4)
5	End of open spooled file	Input	Char(10)
6	Error code	I/O	Char(*)

The Get Spooled File Data (QSPGETSP) API gets data from an existing spooled file. The existing spooled file must have been previously opened by the Open Spooled File (QSPOPNSP) API. Data is retrieved from the existing spooled file by buffers (one or more) and stored in a user space. The data in the user space is used as source to the Put Spooled File Data (QSPPUTSP) API. The number of buffers returned in the user space is no greater than the value specified on the number of buffers to get parameter on the Open Spooled File (QSPOPNSP) API.

| Depending on the data format requested, the actual number
| of bytes returned for each buffer may vary from the size of
| the buffer. Format SPFR0300 may return fewer bytes for
| each buffer than the actual buffer size. Format SPFR0200
| may return more bytes because additional sections, such as
| the buffer information, general information, and page data
| section are included. The buffer size is in relation to the print
| data, not the other information sections.

When creating the user space for the spooled file data, an initial user space size of buffer size multiplied times the number of buffers to get provides adequate space for format SPFR0300. However, this may be too small for format SPFR0200. For format SPFR0200, the size of the user space needed is variable based on the number of pages per buffer. Normally, a good estimate for the user space size needed for SPRF0200 would be the number of buffers multiplied by the buffer size plus 500 bytes. The user space is automatically extended, if necessary, to allow all buffers requested to be stored in the user space.

User spaces can be created using the Create User Space (QUSCRTUS) API. The time it takes to create a user space is decreased if you specify to initialize the space to hexadecimal zeros.

Note: The size of the buffer (currently 512 or 4079 bytes) is determined by the buffer size of the existing spooled file being worked with. The spooled file buffer size is an attribute of the spooled file that can be returned using format SPLA0200 with the QUSRSPLA API.

Authorities and Locks

User Space Authority	*CHANGE
Library Authority	*USE
Output Queue Authority	*USE
User Space Lock	*EXCLRD

Required Parameter Group

Spooled file handle
INPUT; BINARY(4)
The handle returned by the Open Spooled File (QSPOPNSP) API.

User space name and library
INPUT; CHAR(20)
The name of the user space that is to receive the buffer of spooled information. The first 10 characters contain the user space name and the second 10 characters contain the name of the library in which the user space is located. The special values allowed for the library name are *LIBL and *CURLIB. Both user space name and library name are left-justified. If no library is specified as the current library of the job, QGPL is used.

Format name
INPUT; CHAR(8)
The format and the content of the information retrieved from each buffer. The possible values are:

SPFR0100 Information about the print data stored in the buffer.
SPFR0200 Information about the print data stored in the buffer and the print data.
SPRF0300 Print data for the buffer or multiple buffers.

Ordinal number of buffer to be read
INPUT; BINARY(4)
The buffer of the spooled file that is read first (or next).

Any number greater than zero is valid. When a specific buffer is requested, only that buffer is returned. The following special value is supported for this parameter:

-1 Reads the next buffer in the spooled file.

To read sequentially, starting at a specific buffer, the first API call should specify the specific buffer, with subsequent calls of the API specifying the special value -1 (read next). The first API call returns only the specific buffer requested. The subsequent read operations, with -1 specified for the next buffer, return the number of buffers specified on the open operation. When reading an entire spooled file, the special value -1 should be used.

Performance degradation could happen if a specific buffer is always used.

End of open spooled file
INPUT; CHAR(10)
How to handle the situation where the spooled file has not been closed (by this job or another job) and the requested data has not yet been written by the application program. The values supported for this parameter are:

***WAIT** The API waits until the requested data is available or until the spooled file is closed. If the spooled file is closed without the requested data becoming available, the action defined by *ERROR is done.
***ERROR** The API returns either an error code or an error message based on what was specified for the error code parameter.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the User Space

The organization of the user space is dependent on the format (SPFR0100, SPFR0200, or SPRF0300) used.

Figure 56-2 on page 56-21 shows the general structure of the user space for formats SPFR0100 and SPFR0200. If SPFR0100 is used, no print data is returned.

Figure 56-3 on page 56-21 shows the structure for format SPFR0300. If SPRF0300 is used, only the print data is returned, and print data for the buffers requested is returned in one print data section.

Offset values are calculated from the beginning of the user space.

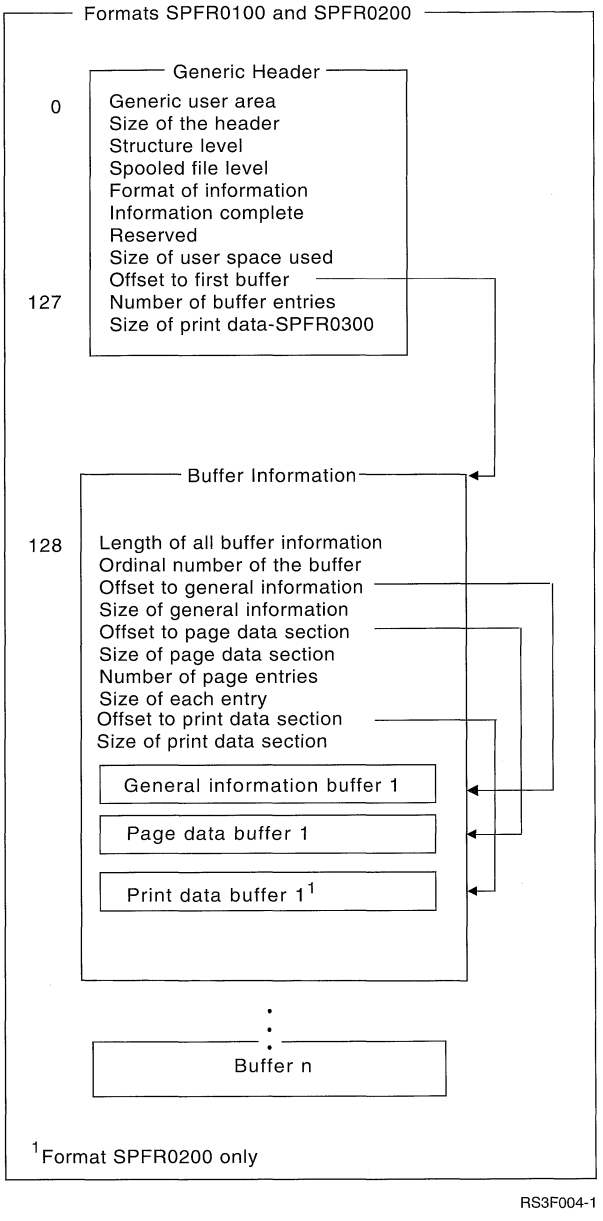


Figure 56-2. Formats SPFR0100 and SPFR0200

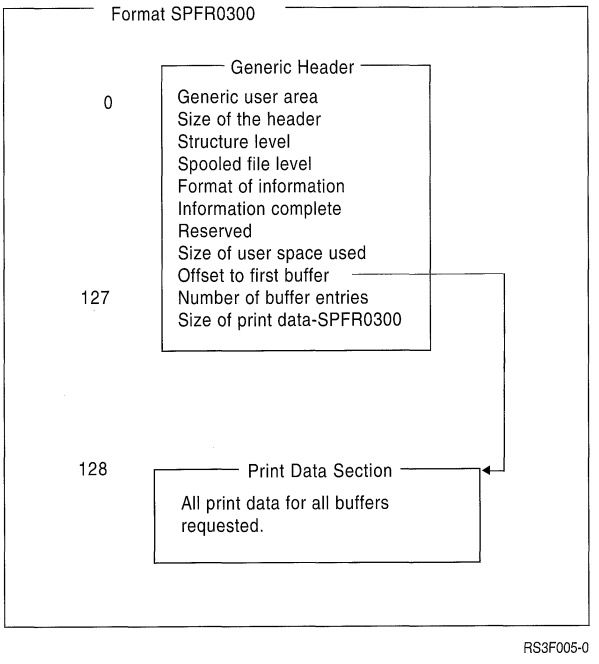


Figure 56-3. Format SPFR0300

Generic Header Section

The following table shows the generic header information returned for the SPFR0100, SPFR0200, and SPRF0300 formats.

For more details about the fields, see "Field Descriptions" on page 56-22.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(64)	Generic user area
64	40	BINARY(4)	Size of header
68	44	CHAR(4)	Structure level
72	48	CHAR(6)	Spooled file level
78	4E	CHAR(8)	Format of the information returned
86	56	CHAR(1)	Information complete indicator
87	57	CHAR(1)	Reserved
88	58	BINARY(4)	Size of user space used
92	5C	BINARY(4)	Offset to first buffer
96	60	BINARY(4)	Number of buffers requested
100	64	BINARY(4)	Number of buffers returned
104	68	BINARY(4)	Size of print data
108	7C	CHAR(20)	Reserved

Field Descriptions

Format of the information returned. The SPFR0100, SPFR0200, or SPFR0300 format of the data returned.

Generic user area. Provided for the user's application program.

Information complete indicator. Used to indicate if all information requested has been supplied.

I Incomplete information. An interruption causes the user space to contain incomplete information about a buffer or buffers. The information for the last buffer is incomplete.

P Partial and accurate information. Partial information is returned when the maximum user space was used and not all of the buffers requested were read. Information about the last buffer is complete.

C Complete and accurate information. All the buffers requested are read and returned. Complete also applies to the case where the end of file was reached on a closed spooled file, but the number of buffers requested was not reached.

Number of buffers requested. The number of buffer entries requested for each read operation.

This is the number specified on the open operation. However, if a specific buffer is requested by the read operation, this value is only one buffer.

Number of buffers returned. The number of buffer entries returned.

This number may be equal to or less than the number of buffers requested. The number may be less because the end of a closed spooled file is reached or the maximum user space has been used before the requested buffer count has been processed.

Offset to first buffer. Locates the data for the first buffer returned. The offset value is from the beginning of the user space. For formats SPFR0100 and SPFR0200, this is the offset to the first buffer information section. For format SPFR0300, this is the offset to the print data section.

Reserved. Used to align a 4-byte boundary.

Size of header. The size of the header section of format SPFR0100, SPFR0200 or SPFR0300 in bytes.

Size of print data. The size of the print data of format SPR0300 in bytes. This is zero for formats SPFR0100 and SPFR0200.

Size of user space used. The number of bytes used (returned) in the user space. This includes the user area, header, general information section, page section, and print data section.

Spooled file level. The level of the spooled file in Version, Release, Modification level format.

Structure level. The level of the structure of the user space. This should always be 0200.

Buffer Information Section

The following buffer information is returned by the QSPGETSP API. For more details about the fields in the following table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Length of all buffer information
4	4	BINARY(4)	Ordinal number of the buffer
8	8	BINARY(4)	Offset to general information section
12	C	BINARY(4)	Size of general information section
16	10	BINARY(4)	Offset to page data section
20	14	BINARY(4)	Size of page data section
24	18	BINARY(4)	Number of page entries
28	1C	BINARY(4)	Size of page entry
32	20	BINARY(4)	Offset to print data section
36	24	BINARY(4)	Size of print data section

Field Descriptions

Length of all buffer information. The size of all information pertaining to this buffer. This includes the general information, page data, and print data.

Number of page entries. The number of page entries in the page data section.

Offset to general information section. The location of the general information section. The offset value is from the beginning of the user space.

Offset to page data section. The location of the page data information section. The offset value is from the beginning of the user space.

Offset to print data section. The location of the print data section. The offset value is from the beginning of the user space.

Ordinal number of the buffer. The ordinal number of the buffer returned.

Size of general information section. The size of the general information section in bytes.

Size of page data section. The size of the page data section in bytes.

Size of page entry. The size of each page entry.

Size of print data section. The size of the print data section in bytes. This is zero (0) for format SPFR0100 because this format does not return this information. For data not previously spooled (creating as opposed to copying) the print data buffer section cannot exceed the size of the spooled file buffer minus the sum of the sizes of the following:

- Constant of 24
- Size of page data section

General Information Section

The general information section of formats SPFR0100 and SPFR0200 has the following structure.

For more details about the fields in the following table, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Nonblank lines in buffer
4	4	BINARY(4)	Nonblank lines in first page
8	8	BINARY(4)	Buffer number of error information
12	C	BINARY(4)	Offset to error recovery information
16	10	BINARY(4)	Size of print data
20	14	CHAR(10)	State
30	1E	CHAR(1)	Last page continues
31	1F	CHAR(1)	Advanced print function file
32	20	CHAR(1)	LAC command array in buffer
33	21	CHAR(1)	Any buffer had LAC
34	22	CHAR(1)	Error recovery information contains LAC
35	23	CHAR(1)	Error recovery information
36	24	CHAR(1)	Zero pages
37	25	CHAR(1)	Load font
38	26	CHAR(1)	IPDS data
39	27	CHAR(5)	Reserved

Field Descriptions

Format SPFR0200 is the format required by the Put Spooled File Data (QSPPUTSP) API. When a spooled file is being copied, the information is retrieved with the QSPGETSP API. When the spooled file is created by a user application, all information in the format must be accurately provided.

Advanced print function file. Whether this spooled file was created by the Advanced Function Printing Utility/400 licensed program. The value is N if the device type is *SCS.

Any buffer had LAC. Whether any buffer had an SCS Load Alternate Characters (LAC) command since last error recovery information was saved. Specify Y for yes and N for no. This should be set to N for all spooled files with a DEVTYPE other than *SCS.

Buffer number of error information. Whether any buffer had an SCS LAC command since the last error recovery information was saved. Specify Y for yes and N for no. This should be set to N for all spooled files with DEVTYPE other than SCS.

Error recovery information. Whether there is error recovery information in this buffer for SCS commands. This error recovery information allows a file to be repositioned and print only some pages of a file, making sure the characters print using the proper code point. Information on the proper code points is stored in LAC commands in the error recovery information. This information is stored for each page that contains SCS commands that can cause characters to print differently based on the code point specified. Specify Y for yes and N for no. This should be set to N for all spooled files with a device type other than SCS.

Error recovery information contains LAC. Error recovery information for SCS commands at the end of this page includes the LAC command found later in the buffer. Specify Y for yes and N for no. This should be set to N for all spooled files with a DEVTYPE other than SCS.

IPDS data. Whether the buffer contains IPDS data. Specify Y for yes and N for no. Buffers of IPDS data may exist in SCS spooled files.

LAC command array in buffer. An LAC command exists in this buffer. Specify Y for yes and N for no. This should be set to N for all spooled files with a DEVTYPE other than SCS.

Last page continues. The last page in this buffer continues in the next buffer when the value is Y.

Load font. Indicates whether a new load font equivalence entry was added to the table for this buffer. Specify Y for yes and N for no. This should be set to N for all spooled files with a device type other than IPDS.

Nonblank lines in buffer. The number of nonblank lines in the buffer.

Nonblank lines in first page. The number of nonblank lines in the first page ending in the buffer.

Offset to error recovery information. Offset to the error recovery information in this buffer. This field is 0 for all spooled files with a DEVTYPE other than SCS.

Reserved. Reserved for byte alignment.

Size of print data. The number of bytes of print data.

State. The state the buffer ends in if the data is IPDS.

**HOME*	Home state
**PAGE*	Page state
**GRAPHICS*	Graphics
**PAGETRANS*	Page transparency
**HOMETRANS*	Home transparency
blank	All non-IPDS files

Zero pages. Whether there are zero (0) pages in the spooled file. Specify Y for yes and N for no.

Page Data Section

Each entry of the page data section of format SPFR0100 and SPFR0200 has the following structure. For more details about the fields in the following table, see "Field Descriptions."

Note: One page data section entry is returned for each page that starts in a returned buffer. The number of pages within a buffer may vary depending on their sizes. Therefore, the number of page data section entries may also vary from buffer to buffer.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Text data start
4	4	BINARY(4)	Any data start
8	8	BINARY(4)	Page offset

Field Descriptions

Any data start. The number of the first line where user data can start on this page. This count includes all data to be printed.

Page offset. The location of the start of this page. The offset value is from the beginning of the print data.

Text data start. Number of the first line where user data can start on this page. This count includes text only.

Print Data Section

The print data section of format SPFR0200 and SPFR0300 has the following structure. For more details about the fields in the following tables, see "Field Descriptions" on page 56-25.

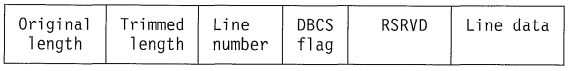
Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(*)	Print data

Print data for format SPFR0300 contains all the print data for all of the buffers returned.

The format of the print text data is dependent on the printer device type.

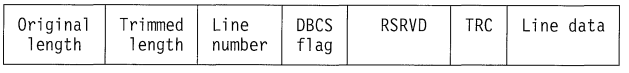
For printer device types of *AFPDS, *AFPDSLINE, and *LINE, the data for a given format may span buffers. The first buffer returned will have one of the following formats:

- Printer device type of *LINE with the control character field set to *NONE and the TRC field set to N.

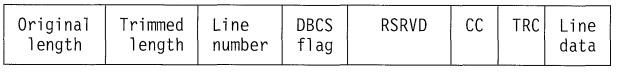


- Printer device type of *LINE with the control character field set to *FCFC and the TRC field set to N.

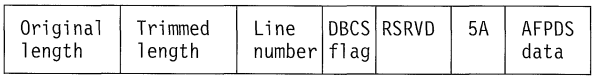
- Printer device type of *LINE with the control character field set to *NONE and the TRC field set to Y.



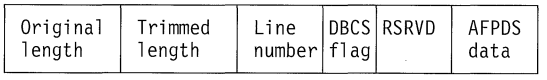
- Printer device type of *LINE with the control character field set to *FCFC and the TRC field set to Y.



- Printer device type of *AFPDS with the control character field set to *FCFC. The TRC field does not apply and should be set to N. The original length field is Y, and the 5A field is Y.



- Printer device type of *AFPDS with the control character field set to *FCFC. The TRC field does not apply and should be set to N. The original length field is Y, and the 5A attribute is N.



- Printer device type of *AFPDS with the control character field set to *FCFC. The TRC field does not apply and should be set to N. The original length field is N, and the 5A field is Y.

5A	AFPDS data

- Printer device type of *AFPDS with the control character field set to *FCFC. The TRC field does not apply and should be set to N. The original length field is N, and the 5A field is N.

AFPDS data

- Printer device type of *AFPDSLINE with the control character field set to *FCFC, the original length field set to Y, the 5A field set to Y, and the TRC field set to N.

  *AFPDSLINE spooled files must have control characters and lengths.

Original length	Trimmed length	line number	DBCS flag	RSRVD	CC	Line data
Original length	Trimmed length	Line number	DBCS flag	RSRVD	5A	AFPDS data

- Printer device type of *AFPDSLINE with the control character field set to *FCFC, the original length field set to Y, the 5A field set to Y, and the TRC field set to Y.

  *AFPDSLINE spooled files must have control characters and lengths.

Original length	Trimmed length	Line number	DBCS flag	RSRVD	CC	TRC	Line data
Original length	Trimmed length	Line number	DBCS flag	RSRVD	5A	AFPDS data	

For a printer device type of *IPDS, the buffer has the following format:

IPDS data

For information on the buffer format of printer device type SCS, see the *Guide to Programming for Printing*.

Field Descriptions

AFPDS data. CHAR(*): A variable length field describing the structured fields that reside in the data stream. See the *Advanced Function Printing: Data Stream Reference* for a description of these structured fields. When the length fields are present, only one structured field can be contained in the AFPDS data.

CC. CHAR(1): Carriage control.

All codes are in hexadecimal notation. The carriage control characters control writing, spacing, and skipping operations as the data is being formatted. It may or may not precede the line data. See the *Print Services Facility User's Programming Guide*, S544-3512, for VM or *Print Services Facility/MVS Applications Programming Guide*, S544-3084, for further information on the valid carriage control codes.

When control characters exist, they can be one of two types: American National Standard printer control characters or machine control characters.

DBCS flag. BIT(1): For DBCS files, whether this page starts in IGC mode. This should be zero for all lines except the first one on the page.

IPDS data. CHAR(*): A variable length field describing the structured fields that reside in the data stream. See the *Intelligent Printer Data Stream Reference*, S544-3417 for a description of these structures.

Line data. CHAR(*): A variable length character field composed of the actual characters to print.

Line number. BIN(15): The line number this page starts on. This should be zero for all lines except the first one on the page.

Original length. BIN(15): The original length of the record before trailing blanks were trimmed. This length does not include the first 8 bytes of the record.

RSRVD. BIT(7): Reserved field.

TRC. CHAR(1): A table-reference character.

All codes are in hexadecimal notation. A table-reference character selects a font to be used when printing the line. See the *Print Services Facility User's Programming Guide for VM*, S544-3512, or *Print Services Facility/MVS Applications Programming Guide*, S544-3084, for further information on the valid table reference codes and how they relate to the CHARS attributes and page definitions. If both CC and TRC are present, CC will be first and the TRC character will be second. The absence or presence of these characters is determined by the control character attribute and TRC attribute.

Trimmed length. BIN(15): The actual or trimmed length of the record. This length does not include the first 8 bytes of the record.

5A. CHAR(1): A special hexadecimal constant used to indicate that a structured field follows.

Restrictions for Print Data Section

Some restrictions apply as to where these record formats can appear in the buffer.

- For *LINE data records it is required that the original length, trimmed length, control character, and TRC character, if they exist, are not split across buffers.
- For *AFPDS data records it is required that the original length, trimmed length, hex 5A control character, and the next 3 bytes of data are not split across a buffer boundary.

For other data types the print text format corresponds to the architecture for that data stream.

Error Messages

CPF24B4	E Severe error while addressing parameter list.
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.
CPF33DF	E Internal data area for opened spooled files destroyed.
CPF33D2	E Spooled file handle not valid.
CPF33D3	E Value &1 not valid for buffer to read parameter.
CPF33D4	E Value &1 not valid for end of open file parameter.
CPF33D5	E Spooled file not opened for operation requested.
CPF33D6	E Buffer &1 not available.
CPF33D7	E Requested number of buffers not returned.
CPF3330	E Necessary resource not available.
CPF811A	E User space &4 in &9 damaged.
CPF9801	E Object &2 in library &3 not found.
CPF9802	E Not authorized to object &2 in &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.
CPF9846	E Error while processing file &1 in library &2.
CPF985{	E Service program &2 for program &3 not found or not available.
CPF9872	E Program &1 in library &2 ended. Reason code &3.

List Spooled Files (QUSLSPL) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	CHAR(20)
2	Format name	Input	CHAR(8)
3	User name	Input	CHAR(10)
4	Qualified output queue name	Input	CHAR(20)
5	Form type	Input	CHAR(10)
6	User-specified data	Input	CHAR(10)

Optional Parameter Group 1:

7	Error code	I/O	CHAR(*)

Optional Parameter Group 2:

8	Qualified job name	Input	CHAR(26)
9	Key for the fields to return	Input	ARRAY(*) of BINARY(4)
10	Number of fields to return	Input	BINARY(4)

The List Spooled Files (QUSLSPL) API is similar to the Work with Spooled Files (WRKSPLF) command or the Work with Job (WRKJOB OPTION(*SPLF)) command. The API generates a list of spooled files on the system and places the list in a user space. The list can include some of the following:

- All spooled files
- Spooled files of specific users or all users
- Spooled files in a specified output queue or in all output queues
- Spooled files for all form types or the standard form type
- Spooled files that have any user-specified data values
- Spooled files generated by a specific job

The generated list replaces any existing list in the user space.

Authorities and Locks

User Space Authority	*CHANGE
Library Authority	*USE
Output Queue Authority	*USE
User Space Lock	*EXCLRD

Required Parameter Group

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 library name.

You can use these special values for the library name:

***CURLIB**	The job's current library
***LIBL**	The library list

Format name
INPUT; CHAR(8)
The format of the spooled file information being returned. You must specify the following:

SPLF0100 Contains the internal identifiers for the spooled file and the job that created the spooled file.

For more information about the SPLF0100 format, see "Format SPLF0100" on page 56-28.

SPLF0200 Information that is shown when the list of the spooled files selected is displayed with the Work with Spooled Files (WRKSPLF) command.

For more information about the SPLF0200 format, see "Format SPLF0200" on page 56-28.

User name
INPUT; CHAR(10)
The name of the user whose spooled files are included in the list. This parameter can be used in conjunction with the output queue and library name, form type, and user-specified data parameters to return a partial list of all the spooled files. The list of spooled files returned is sorted by status, output priority, date, and time. It must be blank if the qualified job name parameter is specified. The possible special values are:

***ALL** Files created by all users
***CURRENT** Files created by the current user

Qualified output queue name
INPUT; CHAR(20)
The name of the output queue whose files are to be included in the list, and the library in which it is located. The first 10 characters contain the output queue name, and the second 10 characters contain the library name. This parameter can be used in conjunction with the user name, form type, and user-specified data parameters to return a partial list of all the spooled files. The list of spooled files returned is sorted by status, output priority, date, and time. It must be blank if the qualified job name parameter is specified.

You can use this special value for the output queue name:

***ALL** Files on all output queues. When you use this value, the library name must be blanks.

You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Form type
INPUT; CHAR(10)
The form type whose files are included in the list. The form type is the value specified on the form type parameter of the printer file. This parameter can be used in conjunction with the user, qualified output queue name, and user-specified data parameters to return a partial list of all the spooled files. The list of spooled files returned is sorted by status, output priority, date, and time. It must be blank if the qualified job name parameter is specified. The special values supported are:

***ALL** Files for all form types
***STD** Only files that specify the standard form type

User-specified data
INPUT; CHAR(10)
The user-specified data value whose files are to be included in the list. This parameter can be used in conjunction with the user name, qualified output queue name, and form type parameters to return a partial list of all the spooled files. The list of spooled files returned is sorted by status, output priority, date, and time. It must be blank if the qualified job name parameter is specified. The special value supported is:

***ALL** Files with any user-specified data values

Optional Parameter Group 1

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Optional Parameter Group 2

Qualified job name
INPUT; CHAR(26)
The qualified job name of the job whose spooled files are to be included in the list. If the user name, the qualified output queue name, the form type, or user-specified data are specified, this parameter must be blank. Otherwise, message CPF34C2 is issued. The list of spooled files returned is sorted by spooled file number.

The qualified job name has three parts:

job name
CHAR(10)
A specific job name, or the following special value:
* Current running job

The rest of the qualified job name parameter must be blank.

user name
CHAR(10)
A specific user profile name, or blank when the job name is asterisk (*).

job number
CHAR(6)
A specific job number, or blank when the job name is asterisk (*).

If this parameter is omitted, the API assumes all blanks.

Keys for the fields to return
INPUT; ARRAY(*) of BINARY(4)
The list of the fields to be returned in the SPLF0200 format. By specifying a set of keys, only the fields whose keys are specified are in the returned format. See "Valid Keys" for a list of valid keys. This field is only used for the SPLF0200 format. This field is ignored if the number of keys for fields to return parameter is 0.

Number of fields to return
INPUT; BINARY(4)
The number of fields to return in the SPLF0200 format. This indicates how many keys are in the array of the keys for the fields to return parameter. This field is only used for the SPLF0200 format, and it must be 0 when the SPLF0100 format is specified.

If this parameter is omitted, the API assumes 0.

Format of the Generated List

The spooled file list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

For details about the user area and generic header fields, see "User Space Format for List APIs" on page 2-7. For details about the remaining items, see the following subtopics. For descriptions of the fields in the list, see "Field Descriptions" on page 56-29.

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
28	1C	CHAR(10)	User name specified
38	26	CHAR(10)	Output queue name specified
48	30	CHAR(10)	Output queue library name specified
58	3A	CHAR(10)	Form type
68	44	CHAR(10)	User-specified data
78	4E	CHAR(26)	Qualified job name specified
104	68	BINARY(4)	Number of key fields specified
108	6C	ARRAY(*) of BINARY(4)	Keys for fields specified

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User name
10	A	CHAR(10)	Output queue name
20	14	CHAR(10)	Output queue library name
30	1E	CHAR(10)	User space name
40	28	CHAR(10)	User space library name
50	32	CHAR(26)	Qualified job name

Format SPLF0100

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User name
10	A	CHAR(10)	Output queue name
20	14	CHAR(10)	Output queue library name
30	1E	CHAR(10)	Form type
40	28	CHAR(10)	User-specified data
50	32	CHAR(16)	Internal job identifier
66	42	CHAR(16)	Internal spooled file identifier

Format SPLF0200

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of fields returned
Offsets vary. These fields repeat, in the order listed, for each key field selected.		BINARY(4)	Length of field information returned
		BINARY(4)	Key field for field returned
		CHAR(1)	Type of data
		CHAR(3)	Reserved
		BINARY(4)	Length of data returned
		CHAR(*)	Data
		CHAR(*)	Reserved

Valid Keys

Key	Type	Field
201	CHAR(10)	Spooled file name
202	CHAR(10)	Job name
203	CHAR(10)	User name
204	CHAR(6)	Job number
205	BINARY(4)	Spooled file number
206	CHAR(10)	Output queue name
207	CHAR(10)	Output queue library name
208	CHAR(10)	Device
209	CHAR(10)	User-specified data
210	CHAR(10)	Status
211	BINARY(4)	Total pages
212	BINARY(4)	Current page
213	BINARY(4)	Copies left to produce
214	CHAR(10)	Form type
215	CHAR(2)	Priority
216	CHAR(7)	Date file was opened
217	CHAR(6)	Time file was opened
218	CHAR(16)	Internal job identifier
219	CHAR(16)	Internal spooled file identifier
220	CHAR(10)	Device type

Field Descriptions

Copies left to produce. The remaining number of copies to be produced on the writer.

Current page. The page number or record number currently being written. The page number may be lower or higher than the page number actually being printed because of buffering done by the system. For example, if the spooled file is routed to a diskette unit or the writer is currently printing job separators or file separators for the spooled file, the page number shown may be zero.

Data. The data returned for the key field.

Date file was opened. The date that the file was opened in the CYYMMDD format.

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

Device. The name of the device on which the spooled file is located. If the spooled file is on an output queue that is the default output queue of a printer, this field contains the name of the output queue. Otherwise, this field is blank.

Device type. The type of device for which the spooled file is intended. The possible values are PRINTER, DISKETTE, or TAPE.

Form type. The type of form to load in the printer to print this file.

Format name. The format of the retrieved output.

Job name. The name of the job that created the spooled file.

Job number. The number of the job that created the spooled file.

Key field for field returned. The identifier of the field returned. See "Valid Keys" on page 56-28 for a list of valid keys. The fields in the list data section are returned in the order specified by the keys for the fields to return parameter for each spooled file in the list.

Keys for fields specified. The list of fields to return in the SPLF0200 format specified in the call to the API.

Internal job identifier. The internal identifier for the job. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Internal spooled file identifier. The input value that other programs use to improve the performance of locating the spooled file on the system. Only the spooled file APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Length of data returned. The length of the data returned for the field.

Length of field information returned. The total length of information returned for this field.

Number of fields returned. This is the number of fields returned to the application.

Number of key fields specified. The number of fields to return in format SPLF0200 specified in the call to the API.

Output queue library name specified. The name of the output queue library specified in the call to the API.

Output queue library name. The library where the output queue is located.

Output queue name specified. The name of the output queue specified in the call to the API.

Output queue name. The name of the output queue in which the file is located.

Priority. The priority of the spooled file. The priority ranges from 1 (highest) to 9 (lowest).

Qualified job name specified. The qualified name of the job specified in the call to the API.

Qualified job name. The qualified name of the job that created the spooled files.

Reserved. The field is reserved.

Spooled file name. The name of the spooled file in the list entry.

Spooled file number. The number of the spooled file in the list entry.

Status. The status of the file. The following list of values is used to describe the status of the file:

**CLOSED* The file has been completely processed by a program, but SCHEDULE(*JOBEND) was specified. The job that produced the spooled file has not finished.
**FINISHED* This spooled file is no longer in the system. These spooled files are included in the list of spooled files only if the qualified job name is specified.
**HELD* The file has been held.
**MESSAGE* This file has a message that needs a reply or needs an action to be taken.
**OPEN* The file has not been completely processed and is not ready to be selected by a writer.
**PENDING* This file is pending (waiting) to be printed.
**PRINTING* The file has been completely sent to the printer, but the print complete status has not been sent back.
**READY* The file is available to be written to an output device by a writer.
**SAVED* The file has been written and then saved. This file remains saved until it is released.
**WRITING* This file is currently being produced by the writer on an output device.

Time file was opened. The time that the file was opened in the HHMMSS format.

HH Hour
MM Minute
SS Second

Total pages. The number of pages this printer file contains.

Type of data. The type of data returned.

C The data is returned in character format.
B The data is returned in binary format

User name. The name of the user that created the spooled file.

User name specified. The name of the user specified in the call to the API.

User space library name. The library containing the user space.

User space library name specified. The library name of the user space specified.

User space name. The name of the user space that contains the format returned.

User space name specified. The name of the user space specified.

User-specified data. The 10 characters of user-specified data that describe the file.

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.
CPF3C30 E Library name &1 is not valid for output queue *ALL
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF3330 E Necessary resource not available.
CPF3336 E Job &5/&4/&3 no longer in the system.
CPF3342 E Job &5/&4/&3 not found.
CPF3343 E Duplicate job names found.
CPF3350 E Job &5/&4/&3 no longer in the system.
CPF34C0 E Value &1 for number of fields to return parameter not valid.
CPF34C1 E Value &1 in keys for fields to return parameter not valid.
CPF34C2 E Too many selection criteria specified.
CPF34C4 E List is too large for user space &1.
CPF811A E User space &4 in &9 damaged.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &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.
CPF9856 E Service program &2 for program &3 not found or not available.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Open Spooled File (QSPOPNSP) API

Parameters

Required Parameter Group:

1	Spooled file handle returned	Output	Binary(4)
2	Qualified job name	Input	Char(26)
3	Internal job identifier	Input	Char(16)
4	Internal spooled file identifier	Input	Char(16)
5	Spooled file name	Input	Char(10)
6	Spooled file number	Input	Binary(4)
7	Number of buffers to get	Input	Binary(4)
8	Error code	I/O	Char(*)

The Open Spooled File (QSPOPNSP) API opens an existing spooled file. After the existing spooled file is opened, the Get Spooled File Data (QSPGETSP) API can then get the data from the file.

Authorities and Locks

Authorities and locks are needed to use the QSPOPNSP API. Following are the details for authorities and locks.

Authorities: The requester is authorized to the spooled file if one or more of the following conditions are met.

- The requester is the owner of the spooled file.
- The requester has *READ authority to the queue on which the spooled file resides, and the queue is specified as DSPDTA(*YES).
- The requester has *SPLCTL authority.
- The requester has *JOBCTL authority, and the queue on which the spooled file resides is specified as OPRCTL(*YES) and DSPDTA(*YES or *NO).
- The requester has ownership of the queue on which the spooled file resides, and the queue is specified as AUTCHK(*OWNER) and DSPDTA(*YES or *NO).
- The requester has read, add, and delete authorities to the queue on which the spooled file resides, and the queue is specified as AUTCHK(*DTAAUT) and DSPDTA(*YES or *NO).

Locks

Spooled File Lock *LSRD

Required Parameter Group

Spooled file handle returned
OUTPUT; BINARY(4)
The handle used on subsequent get (QSPGETSP API) and close (QSPCLOSP API) operations to identify the spooled file.

Qualified job name
INPUT; CHAR(26)
The job that created the spooled file. The qualified job name has three parts:

job name
CHAR(10). A specific job name, or one of the following special values:

- * Only the job that this program is running. The rest of the job name parameter must be blank.
- ***INT** The internal spooled file identifier used to locate the spooled file. The user name and job number must be set to blank.

user name
CHAR(10). A specific user profile name, or blanks when the job name is * or *INT.

job number
CHAR(6). A specific job number, or blanks when the job name is * or *INT.

Internal job identifier
INPUT; CHAR(16)
The internal job identifier for the job that created the spooled file whose attributes are being retrieved. Use the Retrieve Spooled File Attributes (QUSRSPLA) API or one of these APIs to make the identifier available:

"List Spooled Files (QUSLSPL) API" on page 56-26
"List Job (QUSLJOB) API" on page 63-8
"Retrieve Job Information (QUSRJOBI) API" on page 63-24

Internal spooled file identifier
INPUT; CHAR(16)
The internal spooled file identifier for the spooled file whose attributes are being retrieved. To make the identifier available, use the QUSRSPLA API or see "List Spooled Files (QUSLSPL) API" on page 56-26.

Spooled file name
INPUT; CHAR(10)
The name of the spooled file for which you may want to retrieve the attributes. You can use this special value for the name:

- ***INT** The internal spooled file identifier is used to locate the spooled file.

Spooled file number
INPUT; BINARY(4)
The unique number of the spooled file. The valid range is 1 through 9999. The following special values are supported for this parameter:

- **0** Only one spooled file from the job has the specified file name, so the number of the spooled file is not necessary.
- **-1** This uses the highest-numbered spooled file with the specified file name.

Number of buffers to get
INPUT;BINARY(4)
How many buffers to get on each call to the QSPGETSP API. Valid numbers include 1 through 8, 16, 24, and 32. Any values greater than 32 must be a multiple of 32. A value greater than 1 should show some performance

improvement. A value of 8 may give the best performance when using the QSPGETSP API. The following special value is supported for this parameter:

-1 Reads the entire spooled file.

If the user space cannot accommodate all buffers requested, as many complete buffers as possible will be returned in the available space. The information complete indicator in the header section of the data returned is then set to P.

If the end of an open spooled file is encountered before the buffer count is reached, the end of open spooled file parameter on the Get Spooled File Data (QSPGETSP) API determines the action taken.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

How to Select a Spooled File to Be Opened

This table illustrates the valid parameter combinations of qualified job name, internal job identifier, internal spooled file identifier, and spooled file name. The combinations of these parameters identify the spooled file to be opened. For example, when the qualified job name parameter value is *, the internal job identifier must be blank, the internal spooled file identifier must be blank, and the actual name of the spooled file must be given.

Qualified Job Name			Internal Job Identifier	Internal Spooled File Identifier	Spooled File Name
Job Name	User Name	Job Number			
Name	Name	Number	Blanks	Blanks	Name
Name	Blanks	Number	Blanks	Blanks	Name
Name	Name	Blanks	Blanks	Blanks	Name
*	Blanks	Blanks	Blanks	Blanks	Name
*INT	Blanks	Blanks	Internal job identifier	Blanks	Name
*INT	Blanks	Blanks	Internal job identifier	Internal spooled file identifier	*INT

Error Messages

| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3C33 E Spooled file number &1 is not valid.
| CPF3C40 E Spooled file &4 not found.
| CPF3C41 E More than one spooled file with same name.
| CPF3C42 E User name or job number is not blank.
| CPF3C43 E Internal job identifier is not valid.
| CPF3C44 E Internal spooled file identifier is not valid.
| CPF3C58 E Job name specified is not valid
| CPF3C59 E Internal identifier is not blanks and job name is not *INT.
| CPF33C9 E Spooled file name parameter cannot be blank.
| CPF33DA E Value &1 not valid for number of buffers to read parameter.
| CPF33DB E Qualified job name parameter not valid with internal spooled file name.
| CPF33DC E Open or create not valid for diskette files.
| CPF33DD E Maximum number of open spooled files exceeded for this job.
| CPF33DE E Size of internal data for opened spooled file exceeds maximum.
| CPF33DF E Internal data area for opened spooled files destroyed.
| CPF3309 E No files named &1 are active.
| CPF3330 E Necessary resource not available.
| CPF3342 E Job &5/&4/&3 not found.
| CPF3343 E Duplicate job names found.
| CPF3344 E File &1 number &2 no longer in the system.
| CPF3492 E Not authorized to spooled file.
| CPF9838 E User profile storage limit exceeded.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Put Spooled File Data (QSPPUTSP) API

Parameters

Required Parameter Group:

1	Spooled file handle returned	Input	Binary(4)
2	Qualified user space name	Input	Char(20)
3	Error code	I/O	Char(*)

The Put Spooled File Data (QSPPUTSP) API puts data into a spooled file, which was created using the Create Spooled File (QSPCRTSP) API. The data put in the spooled file is taken from a user space. The data in the user space can be created using the Get Spooled File Data (QSPGETSP) API or can be created by a user application.

Before a buffer is put in a spooled file, a limited validity check is performed on the information in the user space for that buffer. The possible errors that result from the values of the fields in the user space can be classified as follows:

- The value with an error can be substituted by the default value. A CPIxxxx message is issued informing the user of the substitution.
- The value with an error causes a CPFxxxx message to be issued. The message is not issued until the validation of the information is complete or a severe error is encountered and validation cannot continue. This pro-

vides the caller with all informational messages as well as the first CPFxxxx message encountered.
- The value with an error causes a CPFxxxx message to be issued immediately.

The buffers must be in the format returned by the Get Spooled File Data (QSPGETSP) API.

Authorities and Locks

User Space Authority *CHANGE
Library Authority *USE
User Space Lock *EXCLRD

Required Parameter Group

Spooled file handle returned
INPUT; BINARY(4)
The handle returned by the QSPCRTSP API.

Qualified user space name
INPUT; CHAR(20)
The name of the user space that contains the buffer of spooled information. The first 10 characters contain the user space name, and the second 10 characters contains the name of the library in which the user space is located. The special values allowed for the library name are *LIBL and *CURLIB. Both entries are left-justified. If no library is specified as the current library of the job, QGPL is used. The format of the user space is the same as that returned by QSPGETSP API. The format specified must be SPFR0200.

To see the format of the user space and how the offset values are calculated, see "Format of the User Space" on page 56-20.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Considerations When Changing or Creating a User Space

When creating your own spooled files or altering the buffers returned by the QSPGETSP API, incorrect data can be put with the QSPPUTSP API. Although errors are caught by the QSPPUTSP API, not all errors are detected.

If the order of the pages, the number of pages, or the number of lines is incorrect, problems can occur when repositioning the spooled file to print it. Repositioning means trying to start printing at a specific page other than page 1. The problems can be caused by:

- Using the restart printing function
- Changing the starting and ending print pages
- Working with an inquiry error message that allows a user to specify what page of the spooled file to restart printing

There are fields in the buffer section of the user space that, if they contain incorrect values, could cause other functions, such as displaying or copying of spooled files, to work incorrectly. These fields are in the general information section and in the page data section.

Fields in the General Information Section

If the state and IPDS data fields contain incorrect or changed values, the Display Spooled File (DSPSPLF) and Copy Spooled File (CPYSPLF) commands can be affected in the following ways:

- State field

  When this field is set to *HOMETRANS or *PAGETRANS, it indicates that the spooled file contains only IPDS transparent data. This field only affects spooled files with a device type of *IPDS.

  A changed or incorrect state field value causes the DSPSPLF and CPYSPLF commands to work the following way:

  - DSPSPLF command

    IPDS transparent data cannot be displayed by the DSPSPLF command. As a result, when the created spooled file is displayed, the data in the buffer with the state field set to *HOMETRANS or *PAGETRANS is not displayed. Furthermore, message CPI3438 (`Intelligent Printer Data Stream (IPDS) data not displayed`) appears. If the spooled file is made up entirely of buffers with the state field set to *HOMETRANS or *PAGETRANS, the spooled file is not displayed. Message CPF3429 (`File cannot be displayed or copied`) is displayed.

  - CPYSPLF command

    IPDS transparent data cannot be copied by the CPYSPLF command. If some buffers of a spooled file have a state field set to *HOMETRANS or *PAGETRANS, those buffers are not copied to the database member. If the spooled file is made up entirely of buffers with the state field set to *HOMETRANS or *PAGETRANS, the spooled file is not copied. Message CPF3429 (`File cannot be displayed or copied`) is displayed.

- IPDS data field

  When this field is set to Y, it indicates that the buffer contains only IPDS data. This field only affects spooled files with a device type of *SCS.

  A changed or incorrect IPDS data field value causes the DSPSPLF and CPYSPLF commands to work the following way:

  - DSPSPLF command

    IPDS data cannot be displayed using the DSPSPLF command. As a result, when a created spooled file is displayed, the data in the buffer with the IPDS

data field set to Y is not displayed. Furthermore, message CPI3437 (Intelligent printer data stream (IPDS) data not displayed) appears at the bottom of the last screen of the DSPSPLF command. If the spooled file is made up entirely of buffers with the IPDS field set to Y, the spooled file is not displayed. Message CPF3429 (File cannot be displayed or copied) is displayed.

- CPYSPLF command

  IPDS data cannot be copied by using the CPYSPLF command. As a result, when a spooled file is copied into the database member, the data in the buffer with the IPDS data field set to Y is not copied. If the spooled file is made up entirely of buffers with the IPDS field set to Y, the spooled file is not copied. Message CPF3429 (File cannot be displayed or copied) is displayed.

Fields in the Page Data Section

If the text data start and page offset fields contain incorrect or changed values, the Display Spooled File (DSPSPLF) command can be affected in the following ways:

- Text data start

  The number of the first line where user data can start on the page. This count includes text only. The text data start field can have an incorrect value when the buffers of the original spooled file are put in the created spooled file in an order other than the original spooled file. This field can also have an incorrect value by simply changing its value to another value higher or lower than was returned when the data was gotten using the QSPGETSP API. This field affects all device types of spooled files.

  A changed or incorrect value for the text data start field causes the DSPSPLF command to work the following way:

  - DSPSPLF command

    When the text data start field contains an incorrect value, DSPSPLF may issue message CPF33F9 (Error occurred while displaying file X number Y) when attempting to find a particular string in the displayed spooled file. Also, the user is able to see only part of the spooled file. The amount of data the user sees depends on the order the buffers were put. For example, if the first buffer of the original spooled file is the last buffer in the created spooled file, the user only sees the pages in the first buffer of the created spooled file. If the second buffer of the original spooled file was the last buffer in the created spooled file, the user sees the pages in the first and second buffer of the created spooled file. The other buffers are there but not displayed.

- Page offset

  The location of the start of this page. The offset value is from the beginning of the print data. This field affects all device types of spooled files.

  A changed or incorrect value for the page offset field causes the DSPSPLF command to work the following way:

  - DSPSPLF command

    When the page offset field contains an incorrect value, DSPSPLF may issue informational message CPI3431 (Line number adjusted). The page information of the individual pages overlaps. The user sees only part of the spooled file because some pages overlap.

Error Messages

CPF24B4	E	Severe error while addressing parameter list.
CPF3CF1	E	Error code parameter not valid.
CPF3C21	E	Format name &1 is not valid.
CPF33DF	E	Internal data area for opened spooled files destroyed.
CPF33D2	E	Spooled file handle not valid.
CPF33D5	E	Spooled file not opened for operation requested.
CPF33F2	E	New page expected at beginning of buffer &1.
CPF33F3	E	Data in buffer &1 exceeds spooled file buffer size.
CPF33F4	E	Beyond end of user space &1 in library &2.
CPF33F6	E	Value in generic header of user space &4 in library &5 not valid.
CPF33F7	E	Value in buffer &6 of user space &4 in library &5 not valid.
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.
CPF9846	E	Error while processing file &1 in library &2.
CPF9856	E	Service program &2 for program &3 not found or not available.
CPF9872	E	Program &1 in library &2 ended. Reason code &3.

Retrieve Output Queue Information (QSPROUTQ) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified output queue name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Output Queue Information (QSPROUTQ) API returns information about the parameters used to create the queue, the current status of the queue, and the number of entries on the queue.

Authorities and Locks

Output queue library authority
The caller needs *READ authority to the output queue library.

Output queue authority
The caller needs one of the following:

- *READ authority to the output queue.
- Job control special authority (*JOBCTL) if the output queue is operator controlled (*OPRCTL(*YES)).
- Spool control special authority (*SPLCTL).

Output queue lock
This API needs an *EXCLRD lock on the output queue.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that receives the information requested.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable provided by the receiver variable parameter. The amount of data returned is truncated if the receiver variable is too small. A length of less than 8 is not valid. If the length specified is greater than the actual length, the results may be unpredictable.

Format name
INPUT; CHAR(8)
The content and format of the queue information being returned. The OUTQ0100 format must be used for the output queue information. See "OUTQ0100 Format" to view the information returned for this format.

Qualified output queue name
INPUT; CHAR(20)
The name of the output queue for which information is returned. The first 10 characters contain the queue name and the second 10 characters contain the name of the library in which the queue resides.

The following special values are supported for the library name:

***LIBL** The library list used to locate the queue.
***CURLIB** The current library for the job is used to locate the queue.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

OUTQ0100 Format

The following table shows the information returned for the OUTQ0100 format. For more details about the fields in the following table see, "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Output queue name
18	12	CHAR(10)	Output queue library name
28	1C	CHAR(10)	Order of files on queue
38	26	CHAR(10)	Display any file
48	30	BINARY(4)	Job separators
52	34	CHAR(10)	Operator controlled
62	3E	CHAR(10)	Data queue name
72	48	CHAR(10)	Data queue library name
82	52	CHAR(10)	Authority to check
92	5C	BINARY(4)	Number of files
96	60	CHAR(10)	Output queue status
106	6A	CHAR(10)	Writer job name
116	74	CHAR(10)	Writer job user name
126	7E	CHAR(6)	Writer job number
136	88	CHAR(10)	Writer job status
146	92	CHAR(10)	Printer device name
156	9C	CHAR(50)	Text description

Field Descriptions

Authority to check. Indicates what type of authorities to the output queue allow the user to control all the files on the queue. The possible values are:

**OWNER* Only the owner of the output queue can control all the output files on the queue.

*DTAAUT Any user with *READ, *ADD, or *DELETE authority to the output queue can control all output files on the queue.

Bytes available. 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.

Data queue library name. The name of the library that contains the data queue.

Data queue name. Name of the data queue associated with this output queue. Whenever a spooled file goes into ready status on the output queue, an entry is placed on the data queue.

*NONE No data queue is associated with this output queue.

Display any file. Whether users who have authority to read this output queue can display the output data of any output file on this queue or only the data in their own files.

*YES Any user having authority to read the queue can display, copy, or send the data of any file on the queue.
*NO Users authorized to use the queue can display, copy, or send the output data of their own files only, unless they have some special authority.
*OWNER Only the owner of a file or a user with *SPLCTL authority can display, copy, send, or move their own spooled files to another output queue.

Job separators. The number of job separators to be placed at the beginning of the output for each job having spooled file entries on this output queue.

0-9 The desired number of job separators.
-2 No job separators are used; instead a message is sent to the writer's message queue at the end of each job indicating that the output can be removed.

Number of files. The number of spooled files that exist on the output queue.

Operator controlled. Whether users with job control authority are allowed to manage or control the files on this queue. Users have job control authority if SPCAUT(*JOBCTL) is specified in their user profile. The possible values are:

*YES Users with job control authority can control the queue and make changes to the files on the queue.
*NO This queue and its entries cannot be controlled or changed by users with job control authority unless they also have some other special authority.

Order of files on queue. The order of the spooled files on the output queue. The possible values are:

*FIFO The queue is first-in first-out for each file. That is, on the queue, new spooled files are placed after all other spooled files that have the same priority.
*JOBNBR The queue entries for the spooled files are sorted in priority sequence using the job number (the date and time that the job entered the system) of the job that created the spooled file.

Output queue library name. The name of the library that contains the output queue.

Output queue name. The name of the output queue.

Output queue status. The status of the output queue. The status may be one of the following values:

RELEASED The queue is released.
HELD The queue is held.

Printer device name. The name of the printer device. If a writer is not started to this queue, this field is blank.

Text description. Text that briefly describes the output queue.

*BLANK There is no text description of the job queue.

Writer job name. The name of the writer job. If a writer job is not started to this queue, this field is blank.

Writer job number. The job number associated with the writer job. If a writer job is not started to this queue, this field is blank.

Writer job status. The status of the writer job. If a writer job is not started to this queue, this field is blank. The status may be one of the following values.

STR The writer job is started to the output queue.
END The writer job is ended.
JOBQ The writer job is on the job queue
HLD The writer job is held.
MSGW The writer job is waiting for a message.

Writer job user name. The name of the user who started the writer job. If a writer job is not started to this queue, this field is blank.

Error Messages

| CPF2207 E Not authorized to use object &1 in library &3 type *&2.
| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3C19 E Error occurred with receiver variable specified.
| CPF3C21 E Format name &1 is not valid.
| CPF3C24 E Length of the receiver variable is not valid.
| CPF3330 E Necessary resource not available.
| CPF3357 E Output queue &1 in library &2 not found.
| CPF8122 E &8 damage on library &4.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Spooled File Attributes (QUSRSPLA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified job name	Input	Char(26)
5	Internal job identifier	Input	Char(16)
6	Internal spooled file identifier	Input	Char(16)
7	Spooled file name	Input	Char(10)
8	Spooled file number	Input	Binary(4)

Optional Parameter:

9	Error code	I/O	Char(*)

The Retrieve Spooled File Attributes (QUSRSPLA) API returns specific information about a spooled file into a receiver variable. The size of the receiver variable determines the amount of information returned. You can specify the spooled file for which information is returned either with the internal job and spooled file identifiers, or with a specific job name, spooled file name, and spooled file number.

For RPG, Pascal, COBOL, and System C/400 PRPQ examples using this API, see Appendix A, "Examples."

Note: This appendix also contains a delete old spooled file (DLTOLDSPLF) example.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The receiver variable that receives the information requested. You can specify the size of the area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results are not predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the information to be returned for the specified spooled file. The valid format names are:

SPLA0100 Basic spooled file attributes. For more information about this format, see "SPLA0100 Format" on page 56-38.

SPLA0200 Detailed spooled file attributes needed to print or duplicate the spooled file. For more information about this format, see "SPLA0200 Format" on page 56-45.

Qualified job name
INPUT; CHAR(26)
The job that created the spooled file. The qualified job name has three parts:

job name CHAR(10). A specific job name, or one of the following special values:
- * Only the job that this program is running. The rest of the job name parameter must be blank.
- *INT The internal spooled file identifier used to locate the spooled file. The user name and job number must be set to blank.

user name CHAR(10). A specific user profile name, or blanks when the job name is * or *INT.

job number CHAR(6). A specific job number, or blanks when the job name is * or *INT.

Internal job identifier
INPUT; CHAR(16)
The internal job identifier for the job that created the spooled file whose attributes are to be retrieved. Use the QUSRSPLA API or one of these APIs to make the identifier available:

- "List Spooled Files (QUSLSPL) API" on page 56-26
- "List Job (QUSLJOB) API" on page 63-8
- "Retrieve Job Information (QUSRJOBI) API" on page 63-24

Internal spooled file identifier
INPUT; CHAR(16)
The internal spooled file identifier for the spooled file whose attributes are retrieved. To make the identifier available, use the QUSRSPLA API or see "List Spooled Files (QUSLSPL) API" on page 56-26.

Spooled file name
INPUT; CHAR(10)
The name of the spooled file for which you may want to retrieve the attributes. You can use this special value for the name:

*INT The internal spooled file identifier is used to locate the spooled file.

Spooled file number
INPUT; BINARY(4)
The unique number of the spooled file. The valid range is 1 through 9999. The following special values are supported for this parameter:

- *0* Only one spooled file from the job has the specified file name, so the number of the spooled file is not necessary.
- *-1* This uses the highest-numbered spooled file with the specified file name.

Note: This parameter must contain a valid value even if the value for the spooled file name parameter is *INT.

A value is required because the QUSRSPLA API performs validity checking on all parameters.

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.

How to Select a Spooled File to Retrieve Its Attributes

This table illustrates the valid parameter combinations of qualified job name, internal job identifier, internal spooled file identifier, and spooled file name. The combinations of these parameters identify the spooled file from which attributes can be retrieved. For example, when the qualified job name parameter value is *, the internal job identifier must be blank, the internal spooled file identifier must be blank, and the actual name of the spooled file must be given.

Qualified Job Name			Internal Job Identifier	Internal Spooled File Identifier	Spooled File Name
Job Name	User Name	Job Number			
Name	Name	Number	Blanks	Blanks	Name
Name	Blanks	Number	Blanks	Blanks	Name
Name	Name	Blanks	Blanks	Blanks	Name
*	Blanks	Blanks	Blanks	Blanks	Name
*INT	Blanks	Blanks	Internal job identifier	Blanks	Name
*INT	Blanks	Blanks	Internal job identifier	Internal spooled file identifier	*INT

SPLA0100 Format

The following table shows the information returned for the SPLA0100 format. For a detailed description of each field, see "Field Descriptions" on page 56-40.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(16)	Internal job identifier
24	18	CHAR(16)	Internal spooled file identifier
40	28	CHAR(10)	Job name
50	32	CHAR(10)	User name
60	3C	CHAR(6)	Job number
66	42	CHAR(10)	Spooled file name
76	4C	BINARY(4)	Spooled file number
80	50	CHAR(10)	Form type
90	5A	CHAR(10)	User-specified data
100	64	CHAR(10)	Status
110	6E	CHAR(10)	File available
120	78	CHAR(10)	Hold file before written
130	82	CHAR(10)	Save file after written
140	8C	BINARY(4)	Total pages
144	90	BINARY(4)	Page or record being written
148	94	BINARY(4)	Starting page
152	98	BINARY(4)	Ending page
156	9C	BINARY(4)	Last page printed
160	A0	BINARY(4)	Restart printing
164	A4	BINARY(4)	Total copies
168	A8	BINARY(4)	Copies left to produce
172	AC	BINARY(4)	Lines per inch
176	B0	BINARY(4)	Characters per inch
180	B4	CHAR(2)	Output priority
182	B6	CHAR(10)	Output queue name
192	C0	CHAR(10)	Output queue library name
202	CA	CHAR(7)	Date file opened
209	D1	CHAR(6)	Time file opened
215	D7	CHAR(10)	Device file name
225	E1	CHAR(10)	Device file library name
235	EB	CHAR(10)	Program that opened file name

Offset		Type	Field
Dec	Hex		
245	F5	CHAR(10)	Program that opened file library name
255	FF	CHAR(15)	Accounting code
270	10E	CHAR(30)	Print text
300	12C	BINARY(4)	Record length
304	130	BINARY(4)	Maximum records
308	134	CHAR(10)	Device type
318	13E	CHAR(10)	Printer device type
328	148	CHAR(12)	Document name
340	154	CHAR(64)	Folder name
404	194	CHAR(8)	System/36 procedure name
412	19C	CHAR(10)	Print fidelity
422	1A6	CHAR(1)	Replace unprintable characters
423	1A7	CHAR(1)	Replacement character
424	1A8	BINARY(4)	Page length
428	1AC	BINARY(4)	Page width
432	1B0	BINARY(4)	Number of separators
436	1B4	BINARY(4)	Overflow line number
440	1B8	CHAR(10)	DBCS data
450	1C2	CHAR(10)	DBCS extension characters
460	1CC	CHAR(10)	DBCS shift-out shift-in (SO/SI) spacing
470	1D6	CHAR(10)	DBCS character rotation
480	1E0	BINARY(4)	DBCS characters per inch
484	1E4	CHAR(10)	Graphic character set
494	1EE	CHAR(10)	Code page
504	1F8	CHAR(10)	Form definition name
514	202	CHAR(10)	Form definition library name
524	20C	BINARY(4)	Source drawer
528	210	CHAR(10)	Printer font
538	21A	CHAR(6)	System/36 spooled file identifier
544	220	BINARY(4)	Page rotation
548	224	BINARY(4)	Justification
552	228	CHAR(10)	Print on both sides (duplex)
562	232	CHAR(10)	Fold records
572	23C	CHAR(10)	Control character
582	246	CHAR(10)	Align forms
592	250	CHAR(10)	Print quality
602	25A	CHAR(10)	Form feed
612	264	CHAR(71)	Volumes (array)
683	2AB	CHAR(17)	File label identifier
700	2BC	CHAR(10)	Exchange type
710	2C6	CHAR(10)	Character code
720	2D0	BINARY(4)	Total records
724	2D4	BINARY(4)	Multiple up (pages per side)
728	2D8	CHAR(10)	Front overlay name
738	2E2	CHAR(10)	Front overlay library name
748	2EC	PACKED(15,5)[1]	Front overlay offset down
756	2F4	PACKED(15,5)[1]	Front overlay offset across
764	2FC	CHAR(10)	Back overlay name
774	306	CHAR(10)	Back overlay library name
784	310	PACKED(15,5)[1]	Back overlay offset down
792	318	PACKED(15,5)[1]	Back overlay offset across
800	320	CHAR(10)	Unit of measure
810	32A	CHAR(10)	Page definition name
820	334	CHAR(10)	Page definition library name
830	33E	CHAR(10)	Line spacing
840	348	PACKED(15,5)[1]	Point size
848	350	PACKED(15,5)[1]	Front margin offset down
856	358	PACKED(15,5)[1]	Front margin offset across
864	360	PACKED(15,5)[1]	Back margin offset down
872	368	PACKED(15,5)[1]	Back margin offset across
880	370	PACKED(15,5)[1]	Length of page
888	378	PACKED(15,5)[1]	Width of page
896	380	CHAR(10)	Measurement method
906	38A	CHAR(1)	Advanced Function Printing (AFP) resource

Offset		Type	Field
Dec	Hex		
907	38B	CHAR(10)	Character set name
917	395	CHAR(10)	Character set library name
927	39F	CHAR(10)	Code page name
937	3A9	CHAR(10)	Code page library name
947	3B3	CHAR(10)	Coded font name
957	3BD	CHAR(10)	Coded font library name
967	3C7	CHAR(10)	DBCS-coded font name
977	3D1	CHAR(10)	DBCS-coded font library name
987	3DB	CHAR(10)	User-defined file

1 COBOL can handle only an 18-digit number. PACKED(15,5) is too big. You must use a 1-character FILLER field followed by a PACKED(13,5) field instead.

Field Descriptions

Accounting code. An identifier assigned by the system to record the resources used to write this file.

Advanced Function Printing (AFP) resource. Whether this spooled file uses AFP resources external to this spooled file, for example, page segments or overlays. The possible values are:

Y The spooled file refers to AFP resources external to the spooled file.
N The spooled file does not refer to AFP resources external to the spooled file.

Align forms. Whether a forms alignment message is sent prior to printing this file. The options are *YES or *NO.

Back margin offset across. For the back side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Back margin offset down. For the back side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are:

-1 The back margin offsets are the same as the front margin offsets.
-2 *DEVD. For printers configured AFP(*YES), no print border is used for back margin offsets across and down. Otherwise, the offset values are 0.
0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Back overlay library name. The name of the library containing the back overlay. The possible values are:

**CURLIB* The current library for the job locates the back overlay.
**LIBL* The library list locates the back overlay.
library name This library is searched for the back overlay.

Back overlay name. The name of the back overlay (the material that prints on the back side of each page). The possible values are:

**FRONTOVL* The back overlay is the same as the front overlay.
**NONE* The file does not use a back overlay.
back overlay name The name of the back overlay.

Back overlay offset across. The offset across from the point of origin where the overlay is printed. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Back overlay offset down. The offset down from the point of origin where the overlay is printed. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Bytes available. The amount of data available to the calling program. In other words, the receiver variable could get this much data from this format if the receiver variable were this large or larger.

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

Character code. Whether the coding for the diskette file is in ASCII or EBCDIC form.

Character set library name. The name of the library containing the font character set object. The possible values are:

**LIBL* The library list is used to locate the font character set object.

library name
This library is searched for the font character set object.

Character set name. The name of the font character set object used to print this file. The possible values are:

**FONT* The information specified on the font parameter is used instead of the character set and code page.
character set name
The name of the font character set object to use.

Characters per inch. The number (in tenths) of characters per horizontal inch, defined in the printer file. The value 100 indicates 10 characters per inch.

Code page. The mapping of graphic characters to code points for this printer. For *DEVD, the system gets the code page from the printer device description.

Code page library name. The name of the library containing the code page used to print this spooled file. The possible values are:

**LIBL*	The library list is used to locate the code page.
library name	This library is searched for the code page name.

Code page name. The name of the code page used to print this spooled file.

Code pages are groups of characters. Within a code page, unique hexadecimal identifiers are assigned to each of the characters.

Coded font library name. The name of the library containing the coded font used to print this spooled file. The possible values are:

**LIBL*	The library list is used to locate the coded font.
library name	This library is searched for the coded font.

Coded font name. The name of the coded font used to print this spooled file. A **coded font** is an AFP resource composed of a character set and a code page. The possible values are:

**FNTCHRSET*
The values used are the values specified on the character set name and library name and code page name and library name fields.
coded font name
The name of the coded font used to print this spooled file.

Control character. Whether this printer file uses the American National Standard printer control character. The possible values are:

**NONE* No print control characters are passed in the data that is printed.
**FCFC* The first character of every record is an American National Standard printer control character.

Copies left to produce. The remaining number of copies to be produced on the printer. Valid values are 1 through 255.

Date file opened. The date that the file was opened in the CYYMMDD format.

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

DBCS character rotation. Whether this printer file causes the double-byte character set (DBCS) characters to be rotated 90 degrees counterclockwise before printing. The possible values are *YES and *NO.

DBCS characters per inch. The number of double-byte characters to be printed per inch. The possible values are:

-1 *CPI: One-half of the characters per inch value.
-2 *CONDENSED: 20 double-byte characters per 3 inches.
5 5 characters per inch.
6 6 characters per inch.
10 10 characters per inch.

DBCS-coded font library name. The name of the library containing the DBCS-coded font used to print this spooled file. The possible values are:

**CURLIB*	The current library is searched for the DBCS-coded font.
**LIBL*	The library list is used to locate the DBCS-coded font.
library name	This library is searched for the DBCS-coded font.

DBCS-coded font name. The name of the DBCS-coded font used to print DBCS-coded data on printers configured as AFP(*YES). The possible values are:

**SYSVAL*
The DBCS-coded font specified in the system value is used.
DBCS-coded font name
The name of the DBCS-coded font being used.

DBCS data. Whether the file can contain double-byte character set (DBCS) data. The options are *YES or *NO.

DBCS extension characters. Whether the system uses the extension character processing function. The possible values are:

**YES* The system processes DBCS extension characters.
**NO* The system does not process DBCS extension characters; it prints extension characters as the undefined character.

DBCS shift-out shift-in (SO/SI) spacing. The presentation of shift-out and shift-in characters when printed. The possible values are:

**YES*	Shift-out and shift-in characters occupy one space.
**NO*	Shift-out and shift-in characters occupy no space.
**RIGHT*	Shift-out characters occupy no space; shift-in characters occupy 2 spaces.

Device file library name. The name of the library that contains the device file.

Device file name. The name of the device file used to create the spooled file.

Device type. The type of device file for which this file is intended. The possible values are PRINTER, DISKETTE, or TAPE.

Document name. The name of the document that was the source of the spooled file.

Ending page. The page at which printing is to end for the file. 0 or 2147483647 indicates the last page. To print the entire file, set this value to 0, 2147483647, or the last page number.

Exchange type. The exchange type of the diskette file. The possible values are:

**STD*	The system allows the exchange type based on the diskette type and sector size.
**BASIC*	The basic exchange type.
**H*	The H exchange type.
**I*	The I exchange type.

File available. The time when this file becomes available to an output device for processing. The possible values are:

**IMMED*	The file is available as soon as the file is opened.
**FILEEND*	The file is available as soon as the file is closed.
**JOBEND*	The file is available when the job that created the file is completed.

File label identifier. The diskette label used when the system last saved the object.

Fold records. Whether records exceeding the printer forms width are folded (wrapped) to the next line. The possible values are *YES or *NO.

Folder name. The name of the folder that contains the source document.

Form definition library name. The name of the library that contains the form definition.

Form definition name. The name of the form definition to use for this print request.

Form feed. The manner in which forms feed to the printer. The possible values are:

**CONT*	Continuous forms
**CUT*	Manually fed cut forms
**AUTOCUT*	Automatically fed cut forms
**DEVD*	As defined in the device description

Form type. The type of form to be loaded in the printer to print this file.

Front margin offset across. For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are:

0–57.79	The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Front margin offset down. For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are:

-2	*DEVD. For printers configured AFP(*YES), no print border is used for front margin offsets across and down. Otherwise, the offset values are 0.
0–57.79	The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Front overlay library name. The name of the library containing the front overlay. The possible values are:

**CURLIB*	The current library for the job locates the front overlay.
**LIBL*	The library list locates the front overlay.
library name	This library is searched for the front overlay.

Front overlay name. The name of the front overlay (the material that prints on the front side of each page). The possible values are:

**NONE* The file does not use the front overlay.

front overlay name
The name of the front overlay.

Front overlay offset across. The offset across from the point of origin where the overlay is printed. The possible values are:

0–57.79	The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Front overlay offset down. The offset down from the point of origin where the overlay is printed. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Graphic character set. The set of graphic characters to be used when printing this file. For *DEVD, the system gets the graphic character set from the printer device description.

Hold file before written. Whether the file is held. The hold parameter handles this function on a Create Printer File (CRTPRTF), Override Printer File (OVRPRTF), or Change Printer File (CHGPRTF) command. The possible values are:

**YES* The file is held.
**NO* The file is not held.

Internal job identifier. The internal identifier for the job. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Internal spooled file identifier. The input value used by other programs to improve the performance of locating the spooled file on the system. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Job name. The name of the job that created the spooled file.

Job number. The number of the job that created the spooled file.

Justification. The percentage that the output is right-justified. The possible values are 100, 50, or 0.

Last page printed. The number of the last printed page in the file if printing ended before the job completed processing.

Length of page. The length of a page. Units of measurement are specified in the measurement method field.

Line spacing. How a file's line data records are spaced when printed. This information is returned only for *LINE and *AFPDSLINE printer device type files. The possible values are:

**SINGLE* Single-spaced
**DOUBLE* Double-spaced
**TRIPLE* Triple-spaced
**CTLCHAR* The control character field determines line spacing. The control character field can have one of these values:
  **NONE* No print control characters are passed in the data that is printed.
  **FCFC* The first character of every record is an American National Standard printer control character.

Lines per inch. The number (in tenths) of lines per vertical inch defined in the printer file. A value of 40 indicates 4 lines per inch.

Maximum records. The maximum number of records allowed in the file at the time the file was opened. A value of 0 indicates no maximum.

Measurement method. The measurement method used for the length of page and width of page fields. The possible values are:

**ROWCOL* Uses rows and columns as the unit of measure.
**UOM* Uses the value specified on the unit of measurement parameter (UOM). UOM is either inches (*INCH) or centimeters (*CM).

Multiple up (pages per side). The number of logical pages that print on each side of each physical page when this file is printed. The possible values are 1, 2, and 4.

Number of separators. The number of file separator pages placed at the beginning of each copy of this file.

Output priority. The priority of the spooled file. The priority ranges from 1 (highest) to 9 (lowest).

Output queue library name. The name of the library that contains the output queue.

Output queue name. The name of the output queue where the file is located.

Overflow line number. The last line to be printed before the data being printed overflows to the next page.

Page definition library name. The name of the library in which the page definition resides. This information is returned only for *LINE or *AFPDSLINE printer device type files.

Page definition name. The name of the page definition to use for the file. This information is returned only for *LINE or *AFPDSLINE printer device type files.

Page length. The page length (in lines per page) used by the spooled file. The valid range is row 1 through 255. The value used should not exceed the actual length of the page used.

Page or record being written. The page number or record number currently being written. The page number may be lower or higher than the page number actually being printed because of buffering done by the system. The page number shown may be zero if:

- The printer file is routed to a diskette unit.
- The writer is currently printing job or file separators for the file.

Page rotation. The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. The possible values are:

-1 *AUTO: Computer output reduction is done automatically if the output is too large to fit on the form, regardless of the print quality.
-2 *DEVD: Page rotation is obtained from the printer device description.
-3 *COR: Output created for a form 13.2 inches wide by 11.0 inches long is adjusted to print on a form 11.0 inches wide by 8.5 inches long.
0 No rotation is done. Printing starts at the edge loaded into the printer first, and is parallel to that edge.
90 Text is rotated 90 degrees clockwise from the 0-degree writing position.
180 Text is rotated 180 degrees clockwise from the 0-degree writing position.
270 Text is rotated 270 degrees clockwise from the 0-degree writing position.

Page width. The page width (in characters per printed line) used by the spooled file. The valid range is column 1 through 378, although some printers have a page width less than 378. The value should not exceed the actual width of the page used.

Point size. The point size in which this file's characters should be printed.

Print fidelity. The kind of error handling that is performed when printing. The possible values are:

**ABSOLUTE* The file is printed only if it can be printed exactly as specified in the data stream.
**CONTENT* The printing overrides errors in the data stream and continues printing with the printer's best quality based on the content fidelity.

Print on both sides (duplex). How the information prints. The possible values are:

**FORMDF* The file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, and *AFPDSLINE printer device type files.
**NO* The printing on the page is on one side only.
**YES* The printing is on both sides of the page with the top of each page the same for both sides.
**TUMBLE* The printing is on both sides with the top of one printed page at the opposite end from the top of the other printed page.

Print quality. The print quality that is used when printing this spooled file. The possible values are:

**STD* Standard
**DRAFT* Draft
**NLQ* Near-letter quality
**FASTDRAFT* Prints at a faster speed than *DRAFT

Print text. The text that is printed at the bottom of each page of printed output and on separator pages.

Printer device type. The type of data stream used to represent the file. The possible values are:

**AFPDS* Advanced Function Printing data stream
**AFPDSLINE* AFPDS data mixed with 1403 line data
**IPDS* Intelligent printer data stream
**LINE* 1403 line data
**SCS* Systems Network Architecture (SNA) character stream
**USERASCII* ASCII data

Printer font. The printer font used. If *DEVD is shown, the printer uses the font defined in the printer device description. If *CPI is shown, the file is printed with a font that has the pitch specified by the CPI (character per inch) field.

Program that opened file library name. The name of the library that contains the program that opened the file.

Program that opened file name. The name of the program that opened the spooled file.

Record length. The length of the file's records. If the field shows -1 (the special value for *RCDFMT), this is an externally defined file and the length is included in the file definition. The length includes the extra length of 1 for carriage controls if it exists.

Replace unprintable characters. Whether characters that cannot be printed are to be replaced with another character. The options are Y (yes) or N (no).

Replacement character. The character that replaces any unprintable characters. This field has a value if the replace unprintables field specifies Y.

Restart printing. The number of the page where printing restarts. When you specify a value while the file is printing, the writer stops printing and restarts on the specified page. If the file is not currently printing, this change takes effect after the first copy prints. The possible values are:

-1 *STRPAGE: The starting page specified in the PAGERANGE parameter is the page on which to restart printing.
-2 *ENDPAGE: Only the last page prints.
-3 *NEXT: The next page of the file to print is the page where printing is restarted.
restart page The page number you specify is where printing restarts.

Save file after written. Whether this file is to be saved after it is written. The possible values are *YES and *NO.

Source drawer. The drawer to be used when the automatically cut form feed option is selected. The possible values are:

1-255 The drawer number.
-1 *E1, the envelope drawer.
-2 *FORMDF, indicating that the file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, or *AFPDSLINE printer device type files.

Spooled file name. The name of the spooled file whose information is retrieved.

Spooled file number. The spooled file number of the specified file.

Starting page. The page at which printing is to start for the file. The possible values are:

0 Printing starts on page 1.
-1 The ending page value.
1 The entire file prints.

Status. The status of the file is one of the following values:

**CLOSED* The program completely processed the file, but SCHEDULE(*JOBEND) was specified, and the job that produced the file has not yet finished.
**HELD* The file is held.
**MESSAGE* The file has a message requiring a reply or an action.
**OPEN* The file was not completely processed and is not ready for a writer.
**PENDING* The file is pending to be printed.
**PRINTER* The complete file was sent to the printer, but a print complete status was not sent back.
**READY* The file is available to be written.
**SAVED* The file was written and is saved. This file remains saved until it is released.
**WRITING* The writer is writing the file on a diskette or a printer device.

System/36 procedure name. The name of the procedure running when the spooled file was created.

System/36 spooled file identifier. The 6-character identifier assigned to the spooled file.

Time file opened. The time that the file was opened in the HHMMSS format where:

HH Hour
MM Minutes
SS Seconds

Total copies. The total number of copies to be produced for this printer file.

Total pages. The number of pages this printer file contains.

Total records. If the spooled file data stream is *AFPDS, *AFPDSLINE, or *LINE or if the spooled file device type is diskette, this field contains the total number of records in the printer file or the diskette file. The field is blank if the file is open.

Unit of measure. The unit of measure to use for specifying distances. The unit of measure is used with certain parameters of the printer file. Page definitions and form definitions are examples of these parameters. The possible values are:

**CM* Centimeters
**INCH* Inches

User name. The name of the user that created the spooled file.

User-defined file. Whether the spooled file was created using an API. The possible values are:

**YES* The spooled file was created using the Create Spooled File (QSPCRTSP) API.
**NO* The spooled file was created by normal system functions.

User-specified data. The 10 characters of user-specified data that describe the file.

Volumes (array). The diskette volumes used for saving the object. The variable returns a maximum of 10 six-character volumes. The volume IDs begin in character positions 1, 8, 15, 22, 29, 36, 43, 50, 57, and 64. If more than 10 volumes are used, a 1 is returned in the 71st character of the variable; otherwise, the 71st character is blank. The variable is returned blank if the object was saved to a save file, or if it was never saved.

Width of page. The width of a page. Units of measurement are specified in the measurement method field.

SPLA0200 Format

The following table shows the information returned for the SPLA0200 format. For more details about the fields in the following table, see "Field Descriptions" on page 56-48.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(8)	Format name
16	10	CHAR(16)	Internal job identifier
32	20	CHAR(16)	Internal spooled file identifier
48	30	CHAR(10)	Job name
58	3A	CHAR(10)	User name
68	44	CHAR(6)	Job number
74	4A	CHAR(10)	Spooled file name
84	54	BINARY(4)	Spooled file number
88	58	CHAR(10)	Form type
98	62	CHAR(10)	User-specified data
108	6C	CHAR(10)	Status
118	76	CHAR(10)	File available
128	80	CHAR(10)	Hold file before written
138	8A	CHAR(10)	Save file after written
148	94	BINARY(4)	Total pages
152	98	BINARY(4)	Page or record being written

Offset		Type	Field
Dec	Hex		
156	9C	BINARY(4)	Starting page
160	A0	BINARY(4)	Ending page
164	A4	BINARY(4)	Last page printed
168	A8	BINARY(4)	Restart printing
172	AC	BINARY(4)	Total copies
176	B0	BINARY(4)	Copies left to produce
180	B4	BINARY(4)	Lines per inch
184	B8	BINARY(4)	Characters per inch
188	BC	CHAR(2)	Output priority
190	BE	CHAR(10)	Output queue name
200	C8	CHAR(10)	Output queue library name
210	D2	CHAR(7)	Date file opened
217	D9	CHAR(6)	Time file opened
223	DF	CHAR(10)	Device file name
233	E9	CHAR(10)	Device file library name
243	F3	CHAR(10)	Program that opened file name
253	FD	CHAR(10)	Program that opened file library name
263	107	CHAR(15)	Accounting code
278	116	CHAR(30)	Print text
308	134	BINARY(4)	Record length
312	138	BINARY(4)	Maximum records
316	13C	CHAR(10)	Device type
326	146	CHAR(10)	Printer device type
336	150	CHAR(12)	Document name
348	15C	CHAR(64)	Folder name
412	19C	CHAR(8)	System/36 procedure name
420	1A4	CHAR(10)	Print fidelity
430	1AE	CHAR(1)	Replace unprintable characters
431	1AF	CHAR(1)	Replacement character
432	1B0	BINARY(4)	Page length
436	1B4	BINARY(4)	Page width
440	1B8	BINARY(4)	Number of separators
444	1BC	BINARY(4)	Overflow line number
448	1C0	CHAR(10)	DBCS data
458	1CA	CHAR(10)	DBCS extension characters

Offset		Type	Field
Dec	Hex		
468	1D4	CHAR(10)	DBCS shift-out shift-in (SO/SI) spacing
478	1DE	CHAR(10)	DBCS character rotation
488	1E8	BINARY(4)	DBCS characters per inch
492	1EC	CHAR(10)	Graphic character set
502	1F6	CHAR(10)	Code page
512	200	CHAR(10)	Form definition name
522	20A	CHAR(10)	Form definition library name
532	214	BINARY(4)	Source drawer
536	218	CHAR(10)	Printer font
546	222	CHAR(6)	System/36 spooled file identifier
552	228	BINARY(4)	Page rotation
556	22C	BINARY(4)	Justification
560	230	CHAR(10)	Print on both sides (duplex)
570	23A	CHAR(10)	Fold records
580	244	CHAR(10)	Control character
590	24E	CHAR(10)	Align forms
600	258	CHAR(10)	Print quality
610	262	CHAR(10)	Form feed
620	26C	CHAR(71)	Volumes (array)
691	2B3	CHAR(17)	File label identifier
708	2C4	CHAR(10)	Exchange type
718	2CE	CHAR(10)	Character code
728	2D8	BINARY(4)	Total records
732	2DC	BINARY(4)	Multiple up (pages per side)
736	2E0	CHAR(10)	Front overlay name
746	2EA	CHAR(10)	Front overlay library name
756	2F4	PACKED(15,5)[1]	Front overlay offset down
764	2FC	PACKED(15,5)[1]	Front overlay offset across
772	304	CHAR(10)	Back overlay name
782	30E	CHAR(10)	Back overlay library name
792	318	PACKED(15,5)[1]	Back overlay offset down
800	320	PACKED(15,5)[1]	Back overlay offset across

Offset		Type	Field
Dec	Hex		
808	328	CHAR(10)	Unit of measure
818	332	CHAR(10)	Page definition name
828	33C	CHAR(10)	Page definition library name
838	346	CHAR(10)	Line spacing
848	350	PACKED(15,5)[1]	Point size
856	358	BINARY(4)	Maximum spooled data record size
860	35C	BINARY(4)	Spooled file buffer size
864	360	CHAR(6)	Spooled file level
870	366	ARRAY(4)	Coded font array
886	376	CHAR(10)	Channel mode
896	380	ARRAY(12)	Channel value array
944	3B0	CHAR(8)	Graphics token
952	3B8	CHAR(10)	Record format
962	3C2	CHAR(2)	Reserved
964	3C4	PACKED(15,5)[1]	Height of drawer 1
972	3CC	PACKED(15,5)[1]	Width of drawer 1
980	3D4	PACKED(15,5)[1]	Height of drawer 2
988	3DC	PACKED(15,5)[1]	Width of drawer 2
996	3E4	BINARY(4)	Number of buffers
1000	3E8	BINARY(4)	Maximum forms width
1004	3EC	BINARY(4)	Alternate forms width
1008	3F0	BINARY(4)	Alternate forms length
1012	3F4	BINARY(4)	Alternate lines per inch
1016	3F8	CHAR(2)	System/38 Text Utility flags
1018	3FA	CHAR(1)	File open
1019	3FB	CHAR(1)	Page count estimated
1020	3FC	CHAR(1)	File stopped on page boundary
1021	3FD	CHAR(1)	TRC for 1403
1022	3FE	CHAR(1)	Define characters
1023	3FF	CHAR(1)	Characters per inch changes
1024	400	CHAR(1)	Transparency
1025	401	CHAR(1)	Double-wide characters
1026	402	CHAR(1)	DBCS character rotation commands

Offset		Type	Field
Dec	Hex		
1027	403	CHAR(1)	Extended code page
1028	404	CHAR(1)	FFT emphasis
1029	405	CHAR(1)	3812 SCS
1030	406	CHAR(1)	Set Line Density command
1031	407	CHAR(1)	Graphics error actions
1032	408	CHAR(1)	5219 commands
1033	409	CHAR(1)	3812 SCS commands
1034	40A	CHAR(1)	Field outlining
1035	40B	CHAR(1)	Final form text
1036	40C	CHAR(1)	Bar code
1037	40D	CHAR(1)	Color
1038	40E	CHAR(1)	Drawer change
1039	40F	CHAR(1)	Character ID
1040	410	CHAR(1)	Lines per inch changes
1041	411	CHAR(1)	Font
1042	412	CHAR(1)	Highlight
1043	413	CHAR(1)	Page rotate
1044	414	CHAR(1)	Subscript
1045	415	CHAR(1)	Superscript
1046	416	CHAR(1)	DDS
1047	417	CHAR(1)	Final form feed
1048	418	CHAR(1)	SCS data
1049	419	CHAR(1)	User-generated data stream
1050	41A	CHAR(1)	Graphics
1051	41B	CHAR(1)	Unrecognizable data
1052	41C	CHAR(1)	ASCII transparency
1053	41D	CHAR(1)	IPDS transparent data
1054	41E	CHAR(1)	OfficeVision/400
1055	41F	CHAR(1)	Lines-per-inch (lpi) value not supported
1056	420	CHAR(1)	CPA3353 message
1057	421	CHAR(1)	Set exception
1058	422	CHAR(1)	Carriage control characters
1059	423	CHAR(1)	Page position
1060	424	CHAR(1)	Character not valid
1061	425	CHAR(1)	Lengths present
1062	426	CHAR(1)	5A present

Offset		Type	Field
Dec	Hex		
1063	427	CHAR(1)	Reserved
1064	428	BINARY(4)	Number of font array entries
1068	42C	BINARY(4)	Number of resource library entries
1072	430	CHAR(1153)	Font equivalence array
2225	8B1	CHAR(631)	Resource library array
2856	B28	CHAR(1)	AS/400-created AFPDS
2857	B28	CHAR(1)	Job character ID specified.
3152	C50	PACKED(15,5)[1]	Front margin offset down
3160	C58	PACKED(15,5)[1]	Front margin offset across
3168	C60	PACKED(15,5)[1]	Back margin offset down
3176	C68	PACKED(15,5)[1]	Back margin offset across
3184	C70	PACKED(15,5)[1]	Length of page
3192	C78	PACKED(15,5)[1]	Width of page
3200	C80	CHAR(10)	Measurement method
3210	C8A	CHAR(1)	Advanced Function Printing (AFP) resource
3211	C8B	CHAR(10)	Character set name
3221	C95	CHAR(10)	Character set library name
3231	C9F	CHAR(10)	Code page name
3241	CA9	CHAR(10)	Code page library name
3251	CB3	CHAR(10)	Coded font name
3261	CBD	CHAR(10)	Coded font library name
3271	CC7	CHAR(10)	DBCS-coded font name
3281	CD1	CHAR(10)	DBCS-coded font library name
3291	CDB	CHAR(10)	User-defined file

1 COBOL can handle only an 18-digit number. PACKED(15,5) is too big. You must use a 1-character FILLER field followed by a PACKED(13,5) field instead.

Field Descriptions

Accounting code. An identifier assigned by the system to record the resources used to write this file.

Advanced Function Printing (AFP) resource. Whether this spooled file refers to AFP resources external to this spooled file, for example, page segments or overlays. The possible values are:

Y The spooled file refers to AFP resources external to the spooled file.
N The spooled file does not refer to AFP resources external to the spooled file.

Align forms. Whether a forms alignment message is sent prior to printing this file. The options are *YES or *NO.

Alternate forms length. The length of the alternate forms in lines. This field applies only to files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields.

Alternate forms width. The width of the alternate forms in character positions. This field applies only to files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields.

Alternate lines per inch. The lines per inch for the alternate forms. This field applies only to files created by the OfficeVision/400 program and should be set to 0 by an application building this format as opposed to retrieving the fields.

AS/400-created AFPDS. Whether the spooled file was created on an AS/400 system using a printer file with the device type (DEVTYPE) parameter value set to *AFPDS.

ASCII transparency. For SCS files, whether ASCII commands are embedded in the ASCII transparency command. The ASCII transparency command is command_ID/command_length/ASCII_data. The command ID is a 1-byte field with value hex 03. The command length is a 1-byte field that contains the length of the command length field and the ASCII data field.

Valid values are Y (yes) or N (no).

Back margin offset across. For the back side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-57). The maximum offset is 57.79 centimeters or 22.75 inches.

Back margin offset down. For the back side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are:

-1 *FRONTMGN. The back margin offsets are the same as the front margin offsets.

-2 *DEVD. For printers configured AFP(*YES), no print border is used for back margin offsets across and down. Otherwise, the offset values are 0.

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-57). The maximum offset is 57.79 centimeters or 22.75 inches.

Back overlay library name. The name of the library containing the back overlay. The possible values are:

**CURLIB* The current library for the job locates the back overlay.

**LIBL* The library list locates the back overlay.

library name This library is searched for the back overlay.

Back overlay name. The name of the back overlay (the material that prints on the back side of each page). The possible values are:

**FRONTOVL*
The back overlay is the same as the front overlay.

**NONE* The file does not use a back overlay.

back overlay name
The name of the back overlay.

Back overlay offset across. The offset across from the point of origin where the overlay is printed. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-57). The maximum offset is 57.79 centimeters or 22.75 inches.

Back overlay offset down. The offset down from the point of origin where the overlay is printed. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-57). The maximum offset is 57.79 centimeters or 22.75 inches.

Bar code. Whether the spooled file contains bar codes created using the BARCODE keyword in the data description specifications (DDS).

For DEVTYPE(*IPDS and *AFPDS), a series of bar code commands is contained in the data stream.

Valid values are Y (yes) or N (no).

Bytes available. The amount of data available to the calling program. In other words, the receiver variable could get this much data from this format if the receiver variable were this large or larger.

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

Carriage control characters. Whether the file has machine carriage control characters.

Valid values are Y (yes) or N (no).

Channel value array. Contains the skip-to line number associated with the channel value. (The first entry in the array applies to channel 1.) The variable returns a maximum of 12 binary(4) values. Each of the values returned is 0 if the mode is *NORMAL.

Channel mode. A variable indicating the channel value mode. The possible values are:

**NORMAL* The normal channel values (1 equals skip to line 1, 2 through B equal space one line before printing, C equals Skip to overflow line).

blank The channel values specified in the channel code array are used.

Character code. Whether the coding for the diskette file is in ASCII or EBCDIC form.

Character ID. Whether the character ID can change within the file.

- For DEVTYPE(*SCS), a Set CGCS through Local ID is contained in the spooled file.
- For DEVTYPE(*IPDS), a Load Font Equivalence command is inserted into the Load Font Equivalence table.
- For DEVTYPE(*AFPDS), a map code font-2 (MCF-2) structured field is contained in the spooled file.

Valid values are Y (yes) or N (no).

Character not valid. Whether incorrect character errors are reported.

Valid values are Y (yes) or N (no).

Character set library name. The name of the library containing the font character set object. The possible values are:

**LIBL* The library list is used to locate the font character set object.

library name This library is searched for the font character set object.

Character set name. The name of the font character set object used to print this file. The possible values are:

**FONT* The information specified on the font parameter is used instead of the character set and code page.

character set name
The name of the font character set object to use.

Characters per inch. The number (in tenths) of characters per horizontal inch, defined in the printer file. The value 100 indicates 10 characters per inch.

For DEVTYPE(*SCS), a Set Character Distance command is contained in the spooled file.

Characters per inch changes. Whether the spooled file changes the characters per inch (cpi) within the file.

For DEVTYPE(*SCS), a Set Character Distance command is contained in the spooled file.

Valid values are Y (yes) or N (no).

Code page. The mapping of graphic characters to code points for this printer. For *DEVD, the system gets the code page from the printer device description.

For DEVTYPE(*SCS), a Set CGCS through GCID command is contained in the spooled file.

Code page library name. The name of the library containing the code page used to print this spooled file. The possible values are:

**LIBL*	The library list is used to locate the code page.
library name	This library is searched for the code page name.

Code page name. The name of the code page used to print this spooled file.

Code pages are groups of characters. Within a code page unique hexadecimal identifiers are assigned to each of the characters.

Coded font array. The name of the fonts used for printing a line data file. This corresponds to the CHARS parameter of MVS and VM printing services. The variable returns a maximum of 4 four-character-coded fonts. The variable is returned blank if no coded fonts are specified. The name specified does not include the two-character prefix of the coded font name (X0 through XG). For more information on the CHARS fields, see the *Print Services Facility/MVS Application Programming Guide* or *Print Services Facility User's Programmers Guide for VM*.

Coded font library name. The name of the library containing the coded font used to print this spooled file. The possible values are:

**LIBL*	The library list is used to locate the coded font.
library name	This library is searched for the coded font.

Coded font name. The name of the coded font used to print this spooled file. A **coded font** is an AFP resource composed of a character set and a code page. The possible values are:

**FNTCHRSET*
: The values used are the values specified on the character set name and library name and code page name and library name fields.

coded font name
: The name of the coded font used to print this spooled file.

Color. Whether an IPDS Write Text command or AFPDS presentation text data (PTX) structured field containing a Set Text Color text control is contained in the spooled file.

For DEVTYPE(*IPDS and *AFPDS), a Set Text Color text control is contained in the spooled file.

Valid values are Y (yes) or N (no).

Control character. Whether this printer file uses the American National Standard printer control character. The possible values are:

**NONE*	No print control characters are passed in the data that is printed.
**FCFC*	The first character of every record is an American National Standard printer control character.

Copies left to produce. The remaining number of copies to be produced on the printer. Valid values are 1 through 255.

CPA3353 message. Whether message CPA3353 is issued when this file is printed.

Valid values are Y (yes) or N (no).

Date file opened. The date that the file was opened in the CYYMMDD format where:

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

DBCS character rotation. Whether this printer file causes the double-byte character set (DBCS) characters to be rotated 90 degrees counterclockwise before printing. The possible values are *YES and *NO.

DBCS character rotation commands. Whether the double byte character set characters are rotated 90 degrees counterclockwise before printing.

- For DEVTYPE(*SCS), a Set Text Orientation command is contained in the spooled file.
- For DEVTYPE(*AFPDS), a map coded font-1 (MCF-1) structured field, that has the character rotation parameter set to hex 8700 in the spooled file.

Valid values are Y (yes) or N (no).

DBCS characters per inch. The number of double-byte characters to be printed per inch. The possible values are:

-1	*CPI: One-half of the characters per inch value.
-2	*CONDENSED: 20 double-byte characters per 3 inches.
5	5 characters per inch.
6	6 characters per inch.
10	10 characters per inch.

DBCS-coded font library name. The name of the library containing the DBCS-coded font. The possible values are:

**CURLIB*	The current library is searched for the DBCS-coded font.

**LIBL*	The library list is used to locate the DBCS-coded font.
library name	This library is searched for the DBCS-coded font.

DBCS-coded font name. The name of the DBCS-coded font used to print DBCS-coded data on printers configured as AFP(*YES). The possible values are:

**SYSVAL*
The DBCS-coded font specified in the system value is used.
DBCS-coded font name
The name of the DBCS-coded font being used.

DBCS data. Whether the file can contain double-byte character set (DBCS) data. The options are *YES or *NO.

DBCS extension characters. Whether the system uses the extension character processing function. The possible values are:

**YES*	The system processes DBCS extension characters.
**NO*	The system does not process DBCS extension characters; it prints extension characters as the undefined character.

DBCS shift-out shift-in (SO/SI) spacing. The presentation of shift-out and shift-in characters when printed. The possible values are:

**YES*	Shift-out and shift-in characters occupy one space.
**NO*	Shift-out and shift-in characters occupy no space.
**RIGHT*	Shift-out characters occupy no space; shift-in characters occupy 2 spaces.

DDS. Whether the file was constructed with DDS.

Valid values are Y (yes) or N (no).

Define characters. Whether the spooled file defines or redefines unique print characters.

For DEVTYPE(*SCS), a Load Alternate Characters command is contained in the spooled file.

Valid values are Y (yes) or N (no).

Device file library name. The name of the library that contains the device file.

Device file name. The name of the device file used to create the spooled file.

Device type. The type of device file for which this file is intended. The possible values are PRINTER, DISKETTE, or TAPE.

Document name. The name of the document that was the source of the spooled file.

Double-wide characters. Whether the spooled file prints everything twice as wide as normal.

For DEVTYPE(*SCS), a Set Font Size command is contained in the data stream.

Valid values are Y (yes) or N (no).

Drawer change. Whether the printer drawer is changed within the spooled file.

- For DEVTYPE(*SCS), a Page Presentation Media command is contained in the spooled file.
- For DEVTYPE(*IPDS), an Execute Order Home State - Select Input Media Source is in the spooled file.
- For DEVTYPE(*AFPDS), a media map and an invoke media map are in the spooled file.

Valid values are Y (yes) or N (no).

Ending page. The page at which printing is to end for the file. 0 or 2147483647 indicates the last page. To print the entire file, set this value to 0, 2147483647, or the last page number.

Exchange type. The exchange type of the diskette file. The possible values are:

**STD*	The system allows the exchange type based on the diskette type and sector size.
**BASIC*	The basic exchange type.
**H*	The H exchange type.
**I*	The I exchange type.

Extended code page. Whether the extended code page changes within the file.

- For DEVTYPE(*SCS), a Set CGCS through GCID is contained in the spooled file.
- For DEVTYPE(*IPDS), a Load Font Equivalence command is inserted into the Load Font Equivalence table.
- For DEVTYPE(*AFPDS), a map code font-2 (MCF-2) structured field is contained in the spooled file.

Valid values are Y (yes) or N (no).

FFT emphasis. Whether the spooled file contains text that is to appear darker than the surrounding text.

For DEVTYPE(*SCS), Begin Emphasis and End Emphasis commands are contained in the spooled file.

Valid values are Y (yes) or N (no).

Field outlining. Whether the spooled file outlines fields of data with boxes. If it does, the file must be printed on a printer that supports field outlining.

For DEVTYPE(*SCS), a Define Grid Line command is contained in the spooled file.

Valid values are Y (yes) or N (no).

File available. The time when this file becomes available to an output device for processing. The possible values are:

**IMMED*	The file is available as soon as the file is opened.
**FILEEND*	The file is available as soon as the file is closed.
**JOBEND*	The file is available when the job that created the file is completed.

File label identifier. The diskette label used when the system last saved the object.

File open. Whether the file is still open when fields are retrieved.

Valid values are Y (yes) or N (no).

File stopped on page boundary. Whether the file has stopped printing on a page boundary.

Valid values are Y (yes) or N (no).

Final form feed. Whether the Final Form Feed command is in the printer file.

Valid values are Y (yes) or N (no).

Final form text. Whether this spooled file contains various functions that are supported on letter-quality printers.

- For DEVTYPE(*SCS), a Document Content Architecture (DCA) command, is contained in the spooled file. DCA commands include:
  - Required New line
  - Required Form Feed
  - Indent Tab
  - Set Presentation Page Size
  - Set Horizontal Margins
  - Set Vertical Margins
  - Release Left Margin
  - Set Line Spacing
  - Set Single Line Distance
  - Justify Text Field Format
  - Set Justify Mode
  - Set Horizontal Tab Stops
  - Set Indent Level
  - Set Exception Action
  - Set Presentation Color
  - Set Spacing Variable
  - Begin and End Overstrike
  - Begin and End Underscore
  - Bell
  - Switch
  - Repeat
  - Tab
  - Backspace
  - Unit Backspace
  - Substitute
  - Word Underscore

Valid values are Y (yes) or N (no).

Fold records. Whether records exceeding the printer forms width are folded (wrapped) to the next line. The possible values are *YES or *NO.

Folder name. The name of the folder that contains the source document.

Font. Whether the spooled file uses multiple fonts.

- For DEVTYPE(*SCS), a Set Font Global command is in the spooled file.
- For DEVTYPE(*IPDS), a Load Font Equivalence command is contained in the Load Font Equivalence table.
- For DEVTYPE(*AFPDS), a map coded font-1 (MCF-1) or map coded font-2 (MCF-2) structured field is in the spooled file.

Valid values are Y (yes) or N (no).

Font equivalence array. The data portion of the Load Font Equivalence (LFE) command for intelligent printer data streams (IPDSs). The variable returns a maximum of 72 sixteen-character font equivalence entries. The current maximum is 48. The additional array space is provided for further expansion. If more than 72 font equivalence entries are used, a 1 is returned in the 1153rd character of the variable; otherwise, the 1153rd character is blank. The format of this entry is described in the *Intelligent Printer Data Stream Reference*.

Form definition library name. The name of the library that contains the form definition.

Form definition name. The name of the form definition to use for this print request.

Form feed. The manner in which forms feed to the printer. The possible values are:

**CONT*	Continuous forms
**CUT*	Manually fed cut forms
**AUTOCUT*	Automatically fed cut forms
**DEVD*	As defined in the device description

Form type. The type of form to be loaded in the printer to print this file.

Format name. The name of the format used to return information.

Front margin offset across. For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far in from the left side of the page printing starts. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-57). The maximum offset is 57.79 centimeters or 22.75 inches.

Front margin offset down. For the front side of a piece of paper, it specifies, in either inches or centimeters (specified in the unit of measure (UOM) parameter), how far down from the top of the page printing starts. The possible values are:

-2 *DEVD. For printers configured AFP(*YES), no print border is used for front margin offsets across and down. Otherwise, the offset values are 0.

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-57). The maximum offset is 57.79 centimeters or 22.75 inches.

Front overlay library name. The name of the library containing the front overlay. The possible values are:

**CURLIB*	The current library for the job locates the front overlay.
**LIBL*	The library list locates the front overlay.
library name	This library is searched for the front overlay.

Front overlay name. The name of the front overlay (the material that prints on the front side of each page). The possible values are:

**NONE* The file does not use the front overlay.
front overlay name
The name of the front overlay.

Front overlay offset across. The offset across from the point of origin where the overlay is printed. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Front overlay offset down. The offset down from the point of origin where the overlay is printed. The possible values are:

0–57.79 The offset in inches or centimeters, according to the unit of measure value (see page 56-45). The maximum offset is 57.79 centimeters or 22.75 inches.

Graphic character set. The set of graphic characters to be used when printing this file. For *DEVD, the system gets the graphic character set from the printer device description.

Graphics. Whether the spooled file contains graphics.

Valid values are Y (yes) or N (no).

Graphics error actions. Whether the file contains graphic error action commands.

For SCS files, the file contains one or more Set Graphic Error Action commands.

Valid values are Y (yes) or N (no).

Graphics token. The printer type on which the graphics in this file can be printed. The possible values are:

4214	Data stream contains graphics for a 4214 printer.
4234	Data stream contains graphics for a 4234 SCS printer.
522X	Data stream contains graphics for a 522X printer.
IPDS	Data stream contains graphics for an IPDS printer.

Height of drawer 1. The height in inches of the paper in drawer 1. This field is for internal use and should be set to 0 by an application building this format as opposed to retrieving the fields.

Height of drawer 2. The height in inches of the paper in drawer 2. This field is for internal use and should be set to 0 by an application building this format as opposed to retrieving the fields.

Highlight. Whether the spooled file contains text that is to appear darker than the surrounding text.

- For DEVTYPE(*SCS), data is overprinted to get the bold effect.
- For DEVTYPE(*IPDS), a Load Font Equivalence command is contained in the Load Font Equivalence table, which has the bold field set on.
- For DEVTYPE(*AFPDS), a map coded font-2 (MCF-2) structured field that has the font weight class parameter set to hex 07 is in the spooled file.

Valid values are Y (yes) or N (no).

Hold file before written. Whether the file is held. The hold parameter handles this function on a Create Printer File (CRTPRTF), Override Printer File (OVRPRTF), or Change Printer File (CHGPRTF) command. The possible values are:

**YES*	The file is held.
**NO*	The file is not held.

Internal job identifier. The internal identifier for the job. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Internal spooled file identifier. The value used as input to other programs to improve the performance of locating the spooled file on the system. Only the OS/400 APIs use this identifier, not any other interface on the system. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

IPDS transparent data. Whether the file contains data from System/36 PRPQs.

Valid values are Y (yes) or N (no).

Job character ID specified. Whether the graphic character set and code page of the spooled file is taken from the coded character set identifier (CCSID) of the job.

Valid values are Y (yes) or N (no).

Job name. The name of the job that created the spooled file.

Job number. The number of the job that created the spooled file.

Justification. The percentage that the output is right-justified. The possible values are 100, 50, or 0.

Last page printed. The number of the last printed page in the file if printing ended before the job completed processing.

Length of page. The length of a page. Units of measurement are specified in the measurement method field.

Lengths present. Whether the 8-byte length information is present in the print text data buffers for format SPLF0200.

Valid values are Y (yes) or N (no).

Line spacing. How a file's line data records are spaced when printed. This information is returned only for *LINE and *AFPDSLINE printer device type files. The possible values are:

**SINGLE* Single-spaced
**DOUBLE* Double-spaced
**TRIPLE* Triple-spaced
**CTLCHAR* The control character field determines line spacing. The control character field can have one of these values:
  **NONE* No print control characters are passed in the data that is printed.
  **FCFC* The first character of every record is an American National Standards printer control character.

Lines per inch. The number (in tenths) of lines per vertical inch defined in the printer file. A value of 40 indicates 4 lines per inch.

Lines per inch changes. Whether the lines per inch (lpi) changes within the spooled file.

- For DEVTYPE(*SCS), a Set Single Line Density command is contained in the spooled file.
- For DEVTYPE(*IPDS), a Load Page Descriptor is contained in the spooled file, which specifies the line-per-inch value
- For DEVTYPE(*AFPDS), a presentation text description (PTD) structured field that specifies how the baseline LPI value changes, is contained in the spooled file.

Valid values are Y (yes) or N (no).

Lines-per-inch (lpi) value not supported. The lines-per-inch (lpi) value is represented in a 1440th-inch value and is not equivalent to 4, 6, 8, 9, or 12. This field applies only to OfficeVision/400 files. It is N for all other spooled files.

Valid values are Y (yes) or N (no).

Maximum forms width. The maximum forms width (in positions) as specified on the printer file.

Maximum records. The maximum number of records allowed in the file at the time the file was opened. A value of 0 indicates no maximum.

Maximum spooled data record size. The length of the largest record in the file. For LINE, AFPDSLINE, and AFPDS files, this length is the length of the largest record in the file. For all other file types, this length is the same as the spooled file buffer size.

Measurement method. The measurement method used for the length of page and width of page fields. The possible values are:

**ROWCOL* Uses rows and columns as the units of measure.
**UOM* Uses the value specified on the unit of measurement parameter (UOM). UOM is either inches (*INCH) or centimeters (*CM).

Multiple up (pages per side). The number of logical pages that print on each side of each physical page when this file is printed. The possible values are 1, 2, and 4.

Number of buffers. The number of buffers currently in the file. If the file is still open when the fields are retrieved, the number of buffers is the number of buffers written to the file thus far.

Number of font array entries. The number of font equivalence array entries used. (The maximum number of font equivalence entries is currently 48.)

Number of resource library entries. The number of entries in the resource library array used. (The maximum number of resource library entries is currently 43.)

Number of separators. The number of file separator pages placed at the beginning of each copy of this file.

OfficeVision/400. Whether the spooled file is generated by the OfficeVision/400 licensed program.

Valid values are Y (yes) or N (no).

Output priority. The priority of the output file. The priority ranges from 1 (highest) to 9 (lowest).

Output queue library name. The name of the library that contains the output queue.

Output queue name. The name of the output queue where the file is located.

Overflow line number. The last line to be printed before the data being printed overflows to the next page.

Page count estimated. Whether the total number of pages is actual or an estimated count.

Valid values are Y (yes) or N (no).

The total page count is estimated in the following situations:

- If the spooled file device type is *AFPDSLINE.
- If the spooled file device type is *LINE and a page definition was used.
- If the spooled file device type is *AFPDS and the spooled file was not created on an AS/400 system.
- If the spooled file device type is *USERASCII.

Page definition library name. The name of the library in which the page definition resides. This information is returned only for *LINE or *AFPDSLINE printer device type files.

Page definition name. The name of the page definition to use for the file. This information is returned only for *LINE or *AFPDSLINE printer device type files.

Page length. The page length (in lines per page) used by the spooled file. The valid range is row 1 through 255. The value used should not exceed the actual length of the page used.

Page or record being written. The page number or record number currently being written. The page number may be lower or higher than the page number actually being printed because of buffering done by the system. The page number shown may be zero if:

- The printer file is routed to a diskette unit.
- The writer is currently printing job or file separators for the file.

Page position. Whether page positioning errors are reported.

Valid values are Y (yes) or N (no).

Page rotate. Whether the spooled file changes the page rotation to be used within the file.

- For DEVTYPE(*SCS), a Set Text Orientation command is contained in the spooled file.
- For DEVTYPE(*IPDS), a Load Page Descriptor command is contained in the spooled file. See the *Intelligent Printer Data Stream Reference* for detailed information about rotating text in a spooled file.
- For DEVTYPE(*AFPDS), a presentation text descriptor (PTD) structured field that specifies the page rotation is contained in the spooled file.

Valid values are Y (yes) or N (no).

Page rotation. The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. The possible values are:

-1	*AUTO: Computer output reduction is done automatically if the output is too large to fit on the form, regardless of the print quality.
-2	*DEVD: Page rotation is obtained from the printer device description.
-3	*COR: Output created for a form 13.2 inches wide by 11.0 inches long is adjusted to print on a form 11.0 inches wide by 8.5 inches long.
0	No rotation is done. Printing starts at the edge loaded into the printer first, and is parallel to that edge.
90	Text is rotated 90 degrees clockwise from the 0-degree writing position.
180	Text is rotated 180 degrees clockwise from the 0-degree writing position.
270	Text is rotated 270 degrees clockwise from the 0-degree writing position.

Page width. The page width (in characters per printed line) used by the spooled file. The valid range is column 1 through 378, although some printers have a page width less than 378. The value should not exceed the actual width of the page used.

Point size. The point size in which this file's characters should be printed.

Print fidelity. The kind of error handling that is performed when printing. The possible values are:

**ABSOLUTE*	The file is printed only if it can be printed exactly as specified in the data stream.
**CONTENT*	The printing overrides errors in the data stream and continues printing with the printers best quality based on the content fidelity.

Print on both sides (duplex). How the information prints. The possible values are:

**FORMDF*	The file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, and *AFPDSLINE printer device type files.
**NO*	The printing on the page is on one side only.
**YES*	The printing is on both sides of the page with the top of each page the same for both sides.
**TUMBLE*	The printing is on both sides with the top of one printed page at the opposite end from the top of the other printed page.

Print quality. The print quality that is used when printing this output. The possible values are:

**STD*	Standard
**DRAFT*	Draft
**NLQ*	Near-letter quality
**FASTDRAFT*	Prints at a faster speed than *DRAFT

Print text. The text that is printed at the bottom of each page of printed output and on separator pages.

Printer device type. The type of data stream used to represent the file. The possible values are:

**AFPDS*	Advanced Function Printing data stream
**AFPDSLINE*	AFPDS data mixed with 1403 line data
**IPDS*	Intelligent printer data stream
**LINE*	1403 line data
**SCS*	Systems Network Architecture (SNA) character stream
**USERASCII*	ASCII data

Printer font. The printer font used. If *DEVD is shown, the printer uses the font defined in the printer device description. If *CPI is shown, the file is printed with a font that has the pitch specified by the CPI (character per inch) field.

Program that opened file library name. The name of the library that contains the program that opened the file.

Program that opened file name. The name of the program that opened the spooled file.

Record format. The format of the records. The possible values are:

**FIXED*	All records in the file are the same size; that is, the original length field of the print data is the same for all records in the file.
**VARIABLE*	Not all records in the file are the same size; that is, the original length field of the print data is the not the same for all records in the file.

Record length. The length of the file's records. If the field shows -1 (the special value for *RCDFMT), this is an externally defined file, and the length is included in the file definition. The length includes the extra length of 1 for carriage controls.

Replace unprintable characters. Whether characters that cannot be printed are to be replaced with another character. The options are Y (yes) or N (no).

Replacement character. The character that replaces any unprintable characters. This field has a value if the replace unprintable characters field specifies Y.

Reserved. An ignored field.

Resource library array. The library names in the library list to use when the spooled file is printed. The variable returns a maximum of 63 ten-character library names. The current maximum is 43. The additional array space is provided for further expansion. If more than 63 resource libraries are used, a 1 is returned in the 631st character of the variable; otherwise, the 631st character is blank.

Restart printing. The number of the page where printing restarts. When you specify a value while the file is printing, the writer stops printing and restarts on the specified page. If the file is not currently printing, this change takes effect after the first copy prints. The possible values are:

-1	*STRPAGE: The starting page specified in the PAGERANGE parameter is the page on which to restart printing.
-2	*ENDPAGE: Only the last page prints.
-3	*NEXT: The next page of the file to print is the page where printing is restarted.
restart page	The page number you specify is where printing restarts.

Save file after written. Whether this file is to be saved after it is written. The possible values are *YES and *NO.

SCS data. Whether the spooled file is created with SCS already in the input data.

Valid values are Y (yes) or N (no).

Set exception. Whether the file has the SCS Set Exception Action (SCS) command in the data stream.

Valid values are Y (yes) or N (no).

Set Line Density command. Whether the lines per inch (lpi) changes within the spooled file or is specified with the Set Line Density SCS command.

For DEVTYPE(*SCS), a Set Line Density command is contained in the spooled file.

Valid values are Y (yes) or N (no).

Source drawer. The drawer to be used when the automatically cut form feed option is selected. The possible values are:

1-255	The drawer number.
-1	*E1, the envelope drawer.
-2	*FORMDF, indicating that the file uses a user-specified form definition. This value is used only for *LINE, *AFPDS, or *AFPDSLINE printer device type files.

Spooled file buffer size. The maximum size of a spooled file buffer. Valid lengths are 512 and 4079.

Spooled file level. The level of the spooled file in Version, Release, and Modification level format (VxRxMx).

Spooled file name. The name of the spooled file whose information is retrieved.

Spooled file number. The spooled file number of the specified file.

Starting page. The page at which printing is to start for the file. The possible values are:

0	Printing starts on page 1.
-1	Use the ending page value.
1	The entire file prints.

Status. The status of the file is one of the following values:

**CLOSED*	The program completely processed the file, but SCHEDULE(*JOBEND) was specified, and the job that produced the file has not yet finished.
**HELD*	The file is held.
**MESSAGE*	The file has a message requiring a reply or an action.
**OPEN*	The file was not completely processed and is not ready for a writer.
**PENDING*	The file is pending to be printed.
**PRINTER*	The complete file was sent to the printer, but a print complete status was not sent back.
**READY*	The file is available to be written.
**SAVED*	The file was written and is saved. This file remains saved until it is released.
**WRITING*	The writer is writing the file on a diskette or a printer device.

Subscript. Whether the spooled file contains subscript.

- For DEVTYPE(*SCS), a Subscript command is contained in the spooled file.
- For DEVTYPE(*IPDS and *AFPDS), a Temporary Baseline Move text control is contained in the spooled file.

Valid values are Y (yes) or N (no).

Superscript. Whether the spooled file contains superscript.

- For DEVTYPE(*SCS), a Superscript command is contained in the spooled file.
- For DEVTYPE(*IPDS and *AFPDS), a Temporary Baseline Move text control is contained in the spooled file.

Valid values are Y (yes) or N (no).

System/36 procedure name. The name of the procedure running when the spooled file was created.

System/36 spooled file identifier. The 6-character identifier assigned to the spooled file.

System/38 text utility flags. The flag for advanced functions.

When you create a file that will contain data that has never been spooled before, set the field to hexadecimal zeros (hex 00).

Time file opened. The time that the file was opened in the HHMMSS format where:

HH Hour
MM Minutes
SS Seconds

Total copies. The total number of copies to be produced for this printer file.

Total pages. The number of pages this printer file contains.

Total records. If the spooled file data stream is *AFPDS, *AFPDSLINE, or *LINE or the spooled file device type is diskette, this field contains the total number of records in the printer file or the diskette file. The field is blank if the file is open.

Transparency. Whether the SCS Transparency command is used in the spooled file. It is set to N for all other device type files, such as IPDS.

For DEVTYPE(*SCS), a TRANSPARENT SCS command is contained in the spooled file.

Valid values are Y (yes) or N (no).

TRC for 1403. Whether the file contains 1403 line data with table-reference characters (TRC). This field is only used for device types *LINE and *AFPDSLINE.

Valid values are Y (yes) or N (no).

Unit of measure. The unit of measure to use for specifying distances. The unit of measure is used with certain parameters of the printer file. Page definitions and form definitions are examples of these parameters. The possible values are:

**CM* Centimeters
**INCH* Inches

Unrecognizable data. Whether the file contains SCS commands that are not valid because of one of the following:

- Command is not recognized
- Command data is not valid
- Command is recognized, but not supported

Valid values are Y (yes) or N (no).

User name. The name of the user that created the spooled file.

User-defined file. Whether the spooled file was created by using an API. The possible values are:

**YES* The spooled file was created using the QSPCRTSP API.
**NO* The spooled file was created by normal system functions.

User-generated data stream. Whether the data stream has been validated by a system program on the AS/400 system when the file was spooled, for example, the OfficeVision/400 program.

Valid values are Y (yes) or N (no).

User-specified data. The 10 characters of user-specified data that describe the file.

Volumes (array). The diskette volumes used for saving the object. The variable returns a maximum of 10 six-character volumes. The volume IDs begin in character positions 1, 8, 15, 22, 29, 36, 43, 50, 57, and 64. If more than 10 volumes are used, a 1 is returned in the 71st character of the variable; otherwise, the 71st character is blank. The variable is returned blank if the object was saved to a save file, or if it was never saved.

Width of drawer 1. The width in inches of the paper in drawer 1. This field is for internal use and should be set to 0 by an application building this format as opposed to retrieving the fields.

Width of drawer 2. The width in inches of the paper in drawer 2. This field is for internal use and should be set to 0 by an application building this format as opposed to retrieving the fields.

Width of page. The width of a page. Units of measurement are specified in the measurement method field.

3812 SCS. Whether the spooled file contains fonts supported only on the 3812 printer.

For DEVTYPE(*SCS), a Set Coded Font Local command with a global ID of greater than 255 is contained in the spooled file.

Valid values are Y (yes) or N (no).

3812 SCS commands. Whether the file contains commands that are not valid on the 3812 printer.

- An incorrect Set Character Set Global ID is found by validity checking the code page with what is valid for the 3812 printer.

- An incorrect Set Character Set Local ID is found by validity checking the code page with what is valid for the 3812 printer.
- A Set Presentation Color command

Valid values are Y (yes) or N (no).

5A present. Whether the 5A hexadecimal constant is present in all AFPDS data for the file (in format SPLF0200).

Valid values are Y (yes) or N (no).

5219 commands. Whether the file contains commands that are not valid on a 5219 printer. The data stream contains one or more of the following SCS commands which, either are not valid on a 5219 or contain parameters that are not valid.

- An incorrect Set Character Set Global ID is found by validity checking the code page with what is valid for the 5219 printer.
- Begin and End Emphasis SCS commands.
- A Set Presentation Page Size command contains a page size value that is not supported.
- A Page Presentation Media command contains a drawer value other than drawer 1 or drawer 2.
- A Set Single Line Distance command contains a distance value that is not supported.
- A Set Single Line Spacing command contains a distance value that is not supported.
- A Justify Text Field Format command contains a value other than 0, 50, or 100 percent.
- A Set Justify Mode Format command contains a value other than 0, 50, or 100 percent.
- A Set Presentation Color command.
- A Begin Overstrike command, which contains an optional CHRID value of other than 0.

Valid values are Y (yes) or N (no).

Error Messages

| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3C19 E Error occurred with receiver variable specified.
| CPF3C20 E Error found by program &1.
| CPD3C21 D Format name &1 is not valid.
| CPD3C24 D Length of the receiver variable is not valid.
| CPD3C40 D Spooled file number &1 is not valid.
| CPD3C42 D User name or job number is not blank.
| CPD3C43 D Internal job identifier is not valid.
| CPD3C44 D Internal spooled file identifier is not valid.
| CPD3C58 D Job name specified is not valid.
| CPD33C9 D Spooled file name parameter cannot be blank.
| CPF3C21 E Format name &1 is not valid.
| CPF3C24 E Length of the receiver variable is not valid.
| CPF3C33 E Spooled file number &1 is not valid.
| CPF3C36 E Number of parameters, &1, entered for this API was not valid.
| CPF3C40 E Spooled file &4 not found.
| CPF3C41 E More than one spooled file with same name.
| CPF3C42 E User name or job number is not blank.
| CPF3C43 E Internal job identifier is not valid.
| CPF3C44 E Internal spooled file identifier is not valid.
| CPF3C58 E Job name specified is not valid
| CPF33C9 E Spooled file name parameter cannot be blank.
| CPF3309 E No files named &1 are active.
| CPF3330 E Necessary resource not available.
| CPF3342 E Job &5/&4/&3 not found.
| CPF3343 E Duplicate job names found.
| CPF3344 E File &1 number &2 no longer in the system.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Writer Information (QSPRWTRI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Receiver variable length	Input	Binary(4)
3	Format name	Input	Char(8)
4	Printer name	Input	Char(10)
5	Error code	I/O	Char(*)

The Retrieve Writer Information (QSPRWTRI) API retrieves writer information associated with the specified printer only when a writer is started to the printer. The information retrieved is similar to what can be seen when running the Work with Writer (WRKWTR) command for a particular printer writer.

Additionally, this API can retrieve information about pending changes to a particular printer writer. The fields that could contain possible pending changes are:

- Next file separators
- Next form type
- Next message option
- Next output queue library name
- Next output queue name
- Next separator drawer

Changes to these fields result when the Change Writer (CHGWTR) command has been used. The value specified in the Changes take effect field determines when the pending changes take place.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The receiver variable that receives the information requested. You can specify the size of the area to be

smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data the area can hold.

Receiver variable length
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 writer information to be returned. The following format name can be used:

WTRI0100 Writer information (See "WTRI0100 Format" for more information.)

Printer name
INPUT; CHAR(10)
The name of the printer for which writer information is to be returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

WTRI0100 Format

The following information is returned for the WTRI0100 format. See "Field Descriptions" for more information.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Started by user
18	12	CHAR(1)	Writing status
19	13	CHAR(1)	Waiting for message status
20	14	CHAR(1)	Held status
21	15	CHAR(1)	End pending status
22	16	CHAR(1)	Hold pending status
23	17	CHAR(1)	Between files status
24	18	CHAR(1)	Between copies status
25	19	CHAR(1)	Waiting for data status
26	1A	CHAR(1)	Waiting for device status
27	1B	CHAR(1)	On job queue status
28	1C	CHAR(4)	Reserved
32	20	CHAR(10)	Writer job name
42	2A	CHAR(10)	Writer job user name
52	34	CHAR(6)	Writer job number
58	3A	CHAR(10)	Printer device type
68	44	BINARY(4)	Number of separators
72	48	BINARY(4)	Drawer for separators
76	4C	CHAR(10)	Align forms
86	56	CHAR(10)	Output queue name
96	60	CHAR(10)	Output queue library name
106	6A	CHAR(1)	Output queue status
107	6B	CHAR(1)	Reserved
108	6C	CHAR(10)	Form type
118	76	CHAR(10)	Message option
128	80	CHAR(10)	Automatically end writer
138	8A	CHAR(10)	Allow direct printing
148	94	CHAR(10)	Message queue name
158	9E	CHAR(10)	Message queue library name
168	A8	CHAR(2)	Reserved
170	AA	CHAR(10)	Changes take effect
180	B4	CHAR(10)	Next output queue name
190	BE	CHAR(10)	Next output queue library name
200	C8	CHAR(10)	Next form type
210	D2	CHAR(10)	Next message option
220	DC	BINARY(4)	Next file separators
224	E0	BINARY(4)	Next separator drawer
228	E4	CHAR(10)	Spooled file name
238	EE	CHAR(10)	Job name
248	F8	CHAR(10)	User name
258	102	CHAR(6)	Job number
264	108	BINARY(4)	Spooled file number
268	10C	BINARY(4)	Page being written
272	110	BINARY(4)	Total pages
276	114	BINARY(4)	Copies left to produce
280	118	BINARY(4)	Total copies

Field Descriptions

Align forms. The time at which the forms alignment message will be sent. Possible values are:

**WTR* The writer determines when the message is sent.
**FILE* Control of the page alignment is specified by each file.

Allow direct printing. Whether the printer writer allows the printer to be allocated to a job that prints directly to a printer.

**NO* Direct printing is not allowed
**YES* Direct printing is allowed.

Automatically end writer. When to end the writer if it is to end automatically.

**NORDYF* When no files are ready to print on the output queue from which the writer is selecting files to be printed.
**FILEEND* When the current spooled file has been printed.
**NO* The writer will not end, but it will wait for more spooled files.

Between copies status. Whether the writer is between copies of a multiple copy spooled file. The possible values are Y (yes) or N (no).

Between files status. Whether the writer is between spooled files. The possible values are Y (yes) or N (no).

Bytes available. Total format data length.

Bytes returned. The length of the data returned.

Changes take effect. The time at which the pending changes to the writer take effect. Possible values are:

**NORDYF* When all the current eligible files are printed.
**FILEEND* When the current spooled file is done printing.
blank No pending changes to the writer.

Copies left to produce. The number of copies left to be printed. This field is set to 0 when no file is printing.

Drawer for separators. Identifies the drawer from which the job and file separator pages are to be taken. Possible values are:

-1 *FILE: The separator page prints from the same drawer that the file is printed from. If you specify a drawer different from the file that contains colored or different type paper, the page separator is more identifiable.
-2 *DEVD: The separator pages will print from the separator drawer specified in the printer device description.
1 The first drawer.
2 The second drawer.
3 The third drawer.

End pending status. Whether an End Writer (ENDWTR) command has been issued for this writer. Possible values are:

N No ENDWTR command was issued.
I *IMMED: The writer ends as soon as its output buffers are empty.
C *CNTRLD: The writer ends after the current copy of the spooled file has been printed.
P *PAGEEND: The writer ends at the end of the page.

Form type. The type of form being used to print the spooled file. Possible values are:

**ALL* The writer is started with the option to print all spooled files of any form type.
**FORMS* The writer is started with the option to print all the spooled files with the same form type before using a different form type.
**STD* The writer is started with the option to print all the spooled files with a form type of *STD.
form type name The writer is started with the option to print all the spooled files with the form type you specified.

Held status. Whether the writer is held. The possible values are Y (yes) or N (no).

Hold pending status. Whether a Hold Writer (HLDWTR) command has been issued for this writer. Possible values are:

N No HLDWTR command was issued.
I *IMMED: The writer is held as soon as its output buffers are empty.
C *CNTRLD: The writer is held after the current copy of the file has been printed.
P *PAGEEND: The writer is held at the end of the page.

Job name. The name of the job that created the spooled file currently being processed by the writer. This field is blank when no spooled file is printing.

Job number. The number of the job that created the spooled file currently being processed by the writer. This field is blank when no spooled file is printing.

Message option. Message option for sending a message to the message queue when this form is finished. Possible values are:

**MSG* A message is sent to the message queue.
**NOMSG* No message is sent to the message queue.
**INFOMSG* An informational message is sent to the message queue.
**INQMSG* An inquiry message is sent to the message queue.

Message queue library name. The name of the library that contains the message queue.

Message queue name. The name of the message queue that this writer uses for operational messages.

Next file separators. The next number of separator pages to be printed when the change to the writer takes place. Possible values are:

-1 *FILE: The number of separator pages is specified by each file.
-10 No pending change to the writer.
number of separators The number of separator pages to be printed.

Next form type. The name of the next form type to be printed. Possible values are:

**ALL* The writer is changed with the option to print all spooled files of any form type.

*FORMS The writer is changed with the option to print all the spooled files with the same form type before using a different form type.
*STD The writer is changed with the option to print all the spooled files with a form type of *STD.
form type name The writer is changed with the option to print all the spooled files with the form type name you specified.
blank No change has been made to this writer.

Next message option. The message option for sending a message to the message queue when the next form type is finished. Possible values are:

*MSG A message is sent to the message queue.
*NOMSG No message is sent to the message queue.
*INFOMSG An informational message is sent to the message queue.
*INQMSG An inquiry message is sent to the message queue.
blank No change pending.

Next output queue library name. The name of the library that contains the next output queue. This field is blank if no changes have been made to the writer.

Next output queue name. The name of the next output queue to be processed. This field is blank if no changes have been made to the writer.

Next separator drawer. This value indicates the drawer from which to take the separator pages if there is a change to the writer. Possible values are:

-1 *FILE: Separator pages print from the same drawer that the spooled file prints from. If you specify a drawer different from the spooled file that contains colored or different type paper, the page separator is more identifiable.
-2 *DEVD: Separator pages print from the separator drawer specified in the printer device description.
-10 No pending change to the writer.
1 The first drawer.
2 The second drawer.
3 The third drawer.

Number of separators. The number of separator pages to be printed. Possible values are:

-1 *FILE: The number of separator pages is specified by each file.
Number of separators The number of separator pages to be printed.

On job queue status. Whether the writer is on a job queue and, therefore, is not currently running. The possible values are Y (yes) or N (no).

Output queue library name. The name of the library that contains the output queue from which spooled files are selected for printing.

Output queue name. The name of the output queue from which spooled files are being selected for printing.

Output queue status. The status of the output queue from which spooled files are being selected for printing. Possible values are:

H The output queue is held.
R The output queue is released.

Page being written. The page number in the spooled file that is currently being processed by the writer. The page number shown may be lower or higher than the actual page number being printed because of buffering done by the system. This field is set to 0 when no spooled file is printing.

Printer device type. The type of the printer being used to print the spooled file. Valid values are:

*SCS SNA (Systems Network Architecture) character stream
*IPDS Intelligent Printer Data Stream

Reserved. An ignored field.

Spooled file name. The name of the spooled file currently being processed by the writer. This field is blank when no spooled file is printing.

Spooled file number. The number of the spooled file currently being processed by the writer. This field is set to 0 when no spooled file is printing.

Started by user. The name of the user that started the writer.

Total copies. The total number of copies to be printed.

Total pages. The total number of pages in the spooled file. Possible values are:

number The number of pages in the spooled file.
0 No spooled file is printing.

User name. The name of the user who created the spooled file currently being processed by the writer. This field is blank when no file is printing.

Waiting for data status. Whether the writer has written all the data currently in the spooled file and is waiting for more data. Possible values are:

N The writer is not waiting for more data.
Y The writer has written all the data currently in the spooled file and is waiting for more data. This condition occurs when the writer is producing an open spooled file with SCHEDULE(*IMMED) specified.

Waiting for device status. Whether the writer is waiting to get the device from a job that is printing directly to the printer.

N The writer is not waiting for the device.
Y The writer is waiting for the device

Waiting for message status. Whether the writer is waiting for a reply to an inquiry message. The possible values are Y (yes) or N (no).

Writer job name. The job name of the printer writer.

Writer job number. The job number of the printer writer.

Writer job user name. The name of the system user.

Writing status. Whether the printer writer is in writing status. The possible values are:

Y The writer is in writing status.
N The writer is not in writing status.
S The writer is writing the file separators.

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.
CPF3C24 E Length of the receiver variable is not valid.
CPF33C8 E No writer information returned for printer &1.
CPF3313 E Writer &1 not active nor on job queue.
CPF3330 E Necessary resource not available.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Transform AFP to ASCII (QWPZTAFP) API

Parameters

Required Parameter Group:

1	Qualified input user space name	Input	Char(20)
2	Length of input data	Input	Binary(4)
3	Qualified output user space name	Input	Char(20)
4	Size of output user space used	Output	Binary(4)
5	Output controls table	Input	Char(*)
6	Length of the output controls table	Input	Binary(4)
7	Error code	I/O	Char(*)

The Transform AFP to ASCII (QWPZTAFP) API transforms an Advanced Function Printing data stream (AFPDS) into an ASCII data stream. The ASCII data stream can be formatted for IBM, Hewlett-Packard, or PostScript-capable printers.

The QWPZTAFP API processes the following AFPDS objects:

- Document
- Presentation page
- Presentation text data
- IM1 image data[1]
- Overlay resources
- Page segment resources

The QWPZTAFP API does not process the following AFPDS objects:

- IO image (IOCA) commands
- Graphic data (GOCA) commands
- Bar code (BCOCA) commands
- Presentation Text 2 (PT2) text commands
- Double-byte character set (DBCS) fonts

AFP commands that are not supported by the QWPZTAFP API are ignored, and no warning or error message is sent to the user.

Fonts and character placement may appear different when printing a transformed data stream on an ASCII printer than if the same data stream (not transformed to ASCII) had been printed on an AFP-capable printer.

Authorities and Locks

Library Authority *USE
User Space Authority *CHANGE, *OBJMGT
User Space Lock *EXCL

Required Parameter Group

Qualified input user space name
INPUT; CHAR(20)
The name of the user space that contains the Advanced Function Printer data stream to be transformed. The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. The special values allowed for the library name are *LIBL and *CURLIB. Both the user space name and the library name are left-justified.

Length of input data
INPUT; BINARY(4)
The number of bytes of AFP data to be transformed in the input space.

Qualified output user space name
INPUT; CHAR(20)
The name of the user space to receive the ASCII data stream. The first 10 characters contain the user space name and the second 10 characters contain the name of the library where the user space is located. The special values allowed for the library name are *LIBL and

[1] IM1 image data is an object format used in the Advanced Function Printing data stream. It is device-dependent and uses pel-to-pel mapping for presentation.

*CURLIB. Both the user space name and the library name are left-justified.

Size of output user space used
OUTPUT; BINARY(4);
The number of bytes written to the output space.

Output controls table
INPUT; CHAR(*)
The format that contains information to control certain aspects of the output data generated by the transform option. See the "Output Controls Table Format" for more information.

Length of the output controls table
INPUT; BINARY(4)
The length of the output controls table 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.

Output Controls Table Format

The following table shows the format for the fields passed to the QWPZTAFP API to control the generated output. See the "Field Descriptions" for more information on the field names in the following table.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Data stream type
4	4	BINARY(4)	Top border size
8	8	BINARY(4)	Left border size
12	C	BINARY(4)	Start page number
16	10	BINARY(4)	End page number
20	14	CHAR(1)	Set character position
21	15	CHAR(1)	Download fonts

Field Descriptions

Data stream type. A number that corresponds to the type of ASCII data stream to be generated. The following table shows the supported values and the data stream type definition.

Figure 56-4. Data Stream Type

Value	Data Stream Type Definition
1	IBM LaserPrinter 4019 in page printer data stream mode
2	IBM LaserPrinter 4029 in page printer data stream mode
3	Hewlett-Packard LaserJet II or compatible (4019 in Hewlett-Packard mode)
4	Hewlett-Packard LaserJet III
5	IBM Personal Page Printer in Hewlett-Packard mode
6	Any printer in PostScript mode
7	Text-only data stream

The default value is 7 for a data stream that is only text.

Download fonts. Allowed values are Y (yes) and N (no).

If Y is used, fonts specified in the AFP data stream are mapped by the transform operation to a scalable font, available on the ASCII printer, or to a font contained in the transform, that can be downloaded and used on the printer. This value is recommended if the AFP data stream uses typographic fonts.

If N is used, fonts specified in the AFP data stream are mapped to a resident font on the ASCII printer. No soft fonts are downloaded. This reduces the size of the output created by the transform operation.

The default value is Y.

Note: Fonts downloaded in Hewlett-Packard printer control language (PCL) format are in portrait-mode only.

End page number. The number of the last page in the input buffer to transform. The following pages are skipped. The special value -1 indicates to transform to the end of the buffer.

The default value is -1.

Left border size. The size in 1/300ths of an inch of the left border. The AFP to ASCII Transform API moves the output towards the right of the page by the specified amount. This can be used to force output outside of the unprintable border of an ASCII printer. This may cause output to be pushed off of the right side of the page.

Any value 0 or greater is valid. The default value is 0.

Set character position. Allowed values are Y (yes) and N (no).

If Y is used, the transform sets the position of each character on the page. This is done to compensate for any difference in character widths between the font specified in the AFP data stream and the font that is used by the transform operation. Using this option improves the appearance of the output in some instances.

If N is used, the transform only sets the position of a character after an absolute move command in the AFP data stream. This reduces the size of the output created by the transform.

The default value is Y.

Start page number. The number of the first page in the input buffer to transform. Previous pages are skipped.

The default value is 1.

Top border size. The size, in 1/300ths of an inch, of the top border. The AFP to ASCII Transform API moves the output down the page by the specified amount. You can use this API to force output outside of the unprintable border of an ASCII printer; however, this may cause some information to print off the page.

Any value 0 or greater is valid. The default value is 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.
CPF6DF0 E ASCII data stream too large for user space &1.
CPF6DF1 E Field number &1 in output controls table not valid.
CPF6DF2 E AFP data stream in user space &1 not valid.
CPF7B10 E Length parameter &1 is not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9856 E Service Program &2 for program &3 not found or not available.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Exit Program for a Customized Separator Page

Parameters

Required Parameter Group:

1	Separator data	Output	Char(*)
2	Separator data length	Input	Binary(4)
3	Separator information	Input	Char(174)
4	Separator information length	Input	Binary(4)

An exit program can be specified on the separator program parameter of the printer device description. If a separator exit program is specified on the printer device description, the system printer writer calls this program to create the separator page. This separator page is used instead of the system separator page.

Required Parameter Group

Separator data
OUTPUT; CHAR(*)
The separator data consists of two parts:

- Options that control how the separator data is to be printed
- User-generated separator data

See the "Separator Data Format" for more information.

Separator data length
INPUT; BINARY(4)
The length of the space used to pass separator data from the exit program.

Separator information
INPUT; CHAR(174)
Information that may be used to generate the separator data.

See the "Separator Information Format" on page 56-66 for more information.

Separator information length
INPUT; BINARY(4)
The length of the space used to pass separator information to the exit program.

Separator Data Format

The following table shows the information passed from the exit program to create the customized separator page for the printer writer. The printer writer takes the information and data, and prints the separator page.

See "Field Descriptions" on page 56-65 for more information.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Transform option
10	A	CHAR(2)	Reserved
12	C	BINARY(4)	Page rotation
16	10	BINARY(4)	Page length
20	14	BINARY(4)	Page width
24	18	BINARY(4)	Lines per inch
28	1C	BINARY(4)	Characters per inch
32	20	BINARY(4)	DBCS Characters per inch
36	24	CHAR(10)	DBCS characters rotation
46	2E	CHAR(10)	Page size measurement method
56	38	CHAR(10)	Print quality
66	42	CHAR(10)	Overlay name
76	4C	CHAR(10)	Overlay library name
86	56	CHAR(98)	Reserved
184	B8	BINARY(4)	Length of user-generated data
188	BC	BINARY(4)	Record length of user data
192	C0	CHAR(*)	User-generated separator data

Field Descriptions

Each field is only valid when the transform option field is specified as *FCFC. Otherwise, the field is ignored.

Characters per inch. The number of characters (in tenths) to be printed per inch. The possible values are:

50 5 characters per inch
100 10 characters per inch
120 12 characters per inch
133 13.3 characters per inch
150 15 characters per inch
167 16.7 characters per inch
180 18 characters per inch
200 20 characters per inch

Any other value, or a value not supported by the particular printer, will result in 10 characters per inch.

DBCS character rotation. Whether double-byte character set (DBCS) characters are to be rotated 90 degrees counter-clockwise before printing. The possible values are *YES and *NO. The default value is *NO.

DBCS characters per inch. The number of double-byte characters to be printed per inch. The possible values are:

-1 One-half of the characters-per-inch value
-2 20 double-byte characters per 3 inches
5 5 characters per inch
6 6 characters per inch
10 10 characters per inch

The default value is *CPI. This is the value specified on the characters per inch parameter in the printer file.

Length of user-generated data. The number of bytes of data (contained in the user-generated separator data field) passed from the user exit program.

The length of the user-generated separator data cannot exceed 8096 bytes for printers configured AFP(*NO). A value greater than 8096 causes the system separator page to be printed.

Lines per inch. The number of lines (in tenths) to print per vertical inch. The possible values are:

30 3 lines per inch
40 4 lines per inch
60 6 lines per inch
75 7.5 lines per inch
80 8 lines per inch
90 9 lines per inch

Any other value, or a value not supported by the particular printer, will result in 6 lines per inch.

This parameter is output from the exit program and is only valid when the Transform Option is specified as *FCFC. Otherwise this parameter is ignored.

Page length. The length of the separator page. The page length can be specified in inches, centimeters, or rows depending on the page size measurement method field.

A zero indicates the page length of the spooled file should be used. A nonzero page length value is valid only when the page width field is also set to a nonzero value. Otherwise, the page length is ignored.

Inches and centimeters are specified in 1/100ths. The value 1100 indicates a page length of 11 inches when the page size measurement method field is *INCH.

Page size measurement method. The measurement method used for the page length and page width fields. The possible values are:

**INCH* The page size is specified in 1/100ths of an inch.
**CM* The page size is specified in 1/100ths of a centimeter.
**ROWCOL* The page size is specified in row and column format.

Page width. The width of the separator page. The page width can be specified in inches, centimeters, or rows depending on the page size measurement method field.

A 0 indicates the page width of the spooled file should be used. A nonzero page width value is valid only when the page length field is also set to a nonzero value. Otherwise, the page width is ignored.

Inches and centimeters are specified in 1/100ths. The value 850 indicates a page width of 8.5 inches when the page size measurement method field is *INCH.

Overlay library name. The name of the library containing the separator overlay. The possible values are:

**LIBL* The library list of the job that created the file locates the overlay.
library name This library is searched for the overlay.

If no value is returned from the exit program, *LIBL is assumed.

Overlay name. The name of the separator overlay (the material that prints on the front side of each separator page). The possible values are:

**NONE* The separator does not use an overlay.
Overlay Name The name of the overlay to print with the separator.

If no value is returned from the exit program, *NONE is assumed. Otherwise this parameter is ignored.

Page rotation. The degree of rotation of the text on the page, with respect to the way the form is loaded into the printer. The possible values are:

0 No rotation is done. Printing starts at the edge loaded into the printer first, and is parallel to that edge.
90 Text is rotated 90-degrees clockwise from the 0-degree writing position.
180 Text is rotated 180-degrees clockwise from the 0-degree writing position.

270 Text is rotated 270-degrees clockwise from the 0-degree writing position.

If no valid value is returned from the exit program, 0 is assumed.

Print quality. The print quality that is used when printing the separator. The possible values are:

**STD* Standard
**DRAFT* Draft
**DEVD* Print quality determined by printer hardware setting
**FASTDRAFT* Fast draft
**NLQ* Near-letter quality

If no valid value is returned from the exit program, *DEVD is assumed. Otherwise this parameter is ignored.

Record length of user data. The length of the *FCFC format records in the user-generated separator data field.

Reserved. An ignored field.

Transform option. Whether the user-generated data is to be transformed into commands recognized by the printer. The possible values are:

**FCFC* The first character of every record is an American National Standards printer control character. The supported control characters and their functions follow:
- *+* Prints record on current line.
- *blank* Moves down 1 line and prints record.
- *0* Moves down 2 lines and prints record.
- *-* Moves down 3 lines and prints record.

All other control character values are treated as blanks and result in the record printing on the next line. Channel values are not supported.

**NONE* User-supplied data is in its final form. No transform operation is done, and the data must contain the appropriate commands corresponding to the printer data stream type.

For the printer data stream type field of *SCS and *IPDS, data stream commands controlling the drawer, rotations, and fonts may be sent to the printer before the user-generated data. This puts the printer in the same state as it is when system default separators are printed. The commands controlling the drawer are based on the SEPDRAWER parameter of the Start Printer Writer (STRPRTWTR) command. The remaining controls are set to defaults or have values set from the spooled file attributes of the spooled file that prints next.

For a printer data stream type of *AFPDS, the initial commands are not sent to the printer and the user must include all commands needed to define an AFPDS document.

User-generated separator data. The data that prints on the separator page. The format of this text is dependent on the transform option and the printer data stream type fields. When *FCFC has been specified for the transform option, the data must be in the *FCFC format. When *NONE is specified for the transform option, the data (text and controls) must be in the format specified by the printer data stream type field.

Separator Information Format

The following table shows the information passed from the printer writer to the exit program. This information is used to create a customized separator page. For more details about the fields in the following table, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(16)	Internal job identifier
16	10	CHAR(16)	Internal spooled file identifier
32	20	CHAR(10)	Job name
42	2A	CHAR(10)	User name
52	34	CHAR(6)	Job number
58	3A	CHAR(10)	Spooled file name
68	44	BINARY(4)	Spooled file number
72	48	CHAR(10)	Printer device name
82	52	CHAR(10)	Printer data stream type
92	5C	CHAR(10)	Type of separator
102	66	CHAR(72)	Reserved

Field Descriptions

Internal job identifier. The internal identifier of the job that owns the spooled file that prints next. Only the OS/400 APIs use this identifier. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Internal spooled file identifier. The internal identifier of the spooled file that prints next. This value is used as input to other programs to improve the performance of locating the spooled file on the system. Only the OS/400 APIs use this identifier. The identifier is not valid following an IPL. If you attempt to use it after an IPL, an exception occurs.

Job name. The name of the job that created the spooled file that prints next.

Job number. The number of the job that created the spooled file that prints next.

Printer device name. The name of the printer device description for the writer that was started.

Printer data stream type. The type of data stream recognized by the printer writer. When the transform option is specified as *NONE, this field must be used to determine the

format of the user-generated separator data field. The possible values are:

*AFPDS	Advanced Function Printing data stream
*IPDS	Intelligent Printer Data Stream
*SCS	Systems Network Architecture (SNA) character stream

Reserved. An ignored field.

Spooled file name. The name of the spooled file that prints next.

Spooled file number. The spooled file number of the spooled file that prints next.

Type of separator. The type of separator you want printed. The possible values are:

*FILE	Separator printed before each copy of a spooled file.
*JOB	Separator printed between spooled files created by different jobs.

User name. The name of the user that created the spooled file that prints next.

Part 22. User Interface APIs

Chapter 57. User Interface APIs

This chapter includes user interface APIs that allow the user to manipulate functions of the interface for an application. These APIs are:

- Display Command Line Window (QUSCMDLN)
- Display Help (QUHDSPH)

Chapter 58, "User Interface Manager APIs" on page 58-1 discusses the user interface management (UIM) APIs.

Display Command Line Window (QUSCMDLN) API

The Display Command Line Window (QUSCMDLN) API displays a window containing a command line. A **window** is an area on the display that is treated as a separate display. Windows have visible boundaries and appear to overlay the display from which they are requested.

From the command line displayed, the user can:

- Enter commands.
- Retrieve commands and run programs.
- Prompt for commands.
- Use help for CL commands.
- Request override commands. When the command line window is removed, all overrides are deleted.

The QUSCMDLN API has no parameters.

For consistency with other AS/400 displays, you should use the F9 key to request a command line window.

The command line window produced by the QUSCMDLN API is shown at the bottom of the following display:

```
CSTINV                    Customer Invoice Menu

Select one of the following:

     1.  Enter or change orders
     2.  Print picking slips
     3.  Release orders
     4.  Print invoices

                                   Command
  ===>
  F4=Prompt   F9=Retrieve   F12=Cancel
```

Display Appearance Problems

There are two rare situations where portions of the underlying display might not be shown correctly while the command line window is shown. The underlying display is not actually changed, and it is shown correctly once the command line window is removed.

The situations are:

1. The underlying display is a text editor or other display that uses special work station printer mode functions. While the command line window is displayed, the special work station printer mode characters are not shown.
2. The underlying display is a bidirectional right-to-left display. While the command line window is displayed, the underlying display is flipped and shown from left-to-right.

Error Messages

CPF9871 E Error occurred while processing.

Display Help (QUHDSPH) API

Parameters

Required Parameter Group:

1	Help identifier array	Input	Char(*)
2	Number of help identifiers	Input	Binary(4)
3	Help type	Input	Array(2) of Binary(4)
4	Full display title	Input	Char(55)
5	Qualified InfoSeeker object name	Input	Char(20)
6	Display type	Input	Char(1)
7	Upper-left corner	Input	Array(2) of Binary(4)
8	Lower-right corner	Input	Array(2) of Binary(4)
9	Cursor location	Input	Array(2) of Binary(4)
10	Error code	I/O	Char(*)

The Display Help (QUHDSPH) API displays help information. The help can be either contextual or extended. It can be formatted as a full display or in a window.

Contextual help provides information about a single item, such as the field on which the user's cursor is positioned when help is requested. **Extended help** provides help for all the items on the display; it contains all contextual help items and can contain additional information as well. A **window** is an area on the display that is treated as a separate display. Windows have visible boundaries and appear to overlay the display from which they are requested.

You can use the QUHDSPH API to handle Help key processing in applications written in data description specifications (DDS) or user-defined data streams (UDDS). You do not need it to handle help requests in applications that use record-level DDS display files with DDS help keywords to present panels because the DDS help keywords handle all help requests.

You do not need to use this API to display help for applications written using panel groups to present panels, because the UIM handles all help requests.

Authorities and Locks

Library Authority	*READ
InfoSeeker Object Authority	*USE
InfoSeeker Object Library Authority	*READ
InfoSeeker Object Lock	*SHRRD
Panel Group Authority	*USE
Panel Group Lock	*SHRRD

Required Parameter Group

Help identifier array
INPUT; CHAR(*)
An array of the help identifiers to display. The list can contain up to 2000 items. Each item has two parts:

Qualified help panel group name
CHAR(20)
The panel group (*PNLGRP) object that contains the help module to display, and the library in which it is located. (A **panel group** is an object with an object type of *PNLGRP. It contains display panels, print panels, or help modules.)

The first 10 characters contain the panel group object name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Help module name
CHAR(32)
The name specified on the NAME attribute of a :HELP. tag in the panel group object. The name must be specified using uppercase, alphabetic characters.

Number of help identifiers
INPUT; BINARY(4)
The number of help identifiers in the help identifier array parameter. The number must be between 1 and 2000.

Help type
INPUT; ARRAY(2) of BINARY(4)
Whether this is a request to display extended or contextual help, and which help identifiers to display for the latter. For contextual help, only a subset of the help identifiers in the help identifier array parameter is initially displayed, and a function key is enabled to display extended help. Extended help includes all help identifiers in the help identifier array parameter, including those used in contextual help.

This parameter is a 2-element array of BINARY(4) values. Both elements are used as indexes into the help identifier array. The array elements are:

1. The first help identifier to display for contextual help. The value must be greater than or equal to 1, and less than or equal to the number of help identifiers in the help identifier array.
2. The last help identifier to display for contextual help. The value must be greater than or equal to the value specified in the first element of this parameter, and less than or equal to the number of help identifiers in the help identifier array.

To display extended help, set the value of the first array element to 1 and the value of the second array element to the value specified in the number of help identifiers parameter.

Full display title
INPUT; CHAR(55)
The default title to use if the help is shown in a full display and if no title is found in the help panel group object.

Qualified InfoSeeker object name
INPUT; CHAR(20)
The InfoSeeker (*SCHIDX) object that can be accessed from the help display, and the library in which it is located. The first 10 characters contain the InfoSeeker object name, and the second 10 characters contain the library name.

You can use the following special value for this parameter:

***NONE** The InfoSeeker function is not made available for this help request. The library name must be blank.

You can use the following special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Display type
INPUT; CHAR(1)
Whether the help information is displayed in a full screen or a window. You must use one of the following values:

Y The help is displayed in a full screen.
N The help can be displayed in a window, depending on the user option (USROPT) value of the user profile.

Upper-left corner
INPUT; ARRAY(2) of BINARY(4)
The upper-left corner of the area on the display for which help is requested. If the help is displayed in a

window, the window is adjacent to the area identified, if possible. The array elements are:

- The row number of the upper-left corner
- The column number of the upper-left corner

Lower-right corner
INPUT; ARRAY(2) of BINARY(4)
The lower-right corner of the area on the display for which help is requested. If the help is displayed in a window, the window is adjacent to the area identified, if possible. The array elements are:

- The row number of the lower-right corner
- The column number of the lower-right corner

Cursor location
INPUT; ARRAY(2) of BINARY(4)
The position of the cursor when help is requested. If the help is displayed in a window, this cursor position is used by the UIM to decide the position and size of the window. The array elements are the row number of the cursor position and the column number of the cursor position.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6E00 E All CPF6Exx messages could be signaled. xx is from 01 to FF.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

Chapter 58. User Interface Manager APIs

The user interface manager (UIM) APIs allow an application developer to manipulate the user interface. These APIs are used in combination with variables, lists, and panel definitions in a panel group object. For more information on creating panel group objects, see the *Guide to Programming Displays*.

The APIs and their parameter descriptions are listed in alphabetical order throughout this chapter. The following list is a task-oriented summary of the categories provided by the UIM:

- Application APIs:

  Close Application (QUICLOA)
  Open Display Application (QUIOPNDA)
  Open Print Application (QUIOPNPA)

- Display APIs:

  Add Pop-Up Window (QUIADDPW)
  Display Panel (QUIDSPP)
  Remove Pop-up Window (QUIRMVPW)
  Set Screen Image (QUISETSC)

- List APIs:

  Add List Entry (QUIADDLE)
  Add List Multiple Entries (QUIADDLM)
  Delete List (QUIDLTL)
  Get List Entry (QUIGETLE)
  Get List Multiple Entry (QUIGETLM)
  Remove List Entry (QUIRMVLE)
  Retrieve List Attributes (QUIRTVLA)
  Set List Attributes (QUISETLA)
  Update List Entry (QUIUPDLE)

- Print APIs:

  Add Print Application (QUIADDPA)
  Print Panel (QUIPRTP)
  Remove Print Application (QUIRMVPA)

- Variable pool APIs:

  Get Dialog Variable (QUIGETV)
  Put Dialog Variable (QUIPUTV)

| ***Passing Arguments:*** When calling UIM APIs, the calling
| program must pass arguments by reference. That is, all UIM
| API programs expect the calling program to pass a space pointer to each of the argument values. For some HLL compilers, this is the only way of passing arguments when calling a program.

Unless otherwise noted, all argument values must be in uppercase, and left-adjusted with trailing blanks. For example, if a program accepts a special value of "ALL" for a parameter defined as CHAR(4), "ALL " must be passed (without the quotes).

Terms and Definitions

Application variable pool. The set of all dialog variable values for an open application.

Argument list. In UIM, this list consists of values that are passed to a program.

| **Coded character set identifier (CCSID)**. A 16-bit number
| identifying a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other relevant information that uniquely identifies the coded graphic character representation used.

Contextual help. Help information about a single item, such as the field on which the cursor is positioned when help is requested.

Dialog variable. A named element in a panel group used to pass data values between programs or between a program and a user. The current contents of all dialog variables corresponding to variables in the list are saved as each entry is added.

Error variable. The dialog variable specified on the ERRVAR attribute of the variable definition (VAR) tag. The error variable is used to set and test the error status of the dialog variable named on the NAME attribute of the VAR tag.

| **Extended action entry**. The first line below the column
| headings in a list. This line contains entry-capable fields for the option column and for at least one additional list column.

Extended action list area. A list area of a panel that contains an extended action entry.

Extended help. Help information for all the items on the display; it contains all contextual help items and can contain additional information as well.

List entry handle. A value that uniquely distinguishes an entry in a UIM list until it is removed from the list. A list entry handle is meaningful only for a particular open application, list, and entry combination. It has no meaning in any other open UIM application, or even in the same application if the list or the entry is deleted and then re-created. Unpredictable results are possible if a list entry handle is used outside of this definition.

Message reference key. A unique string of characters that identify a particular instance of a message in a queue. The message key can also be used to refer to a specific instance of a message in order to move, receive, reply to, resend, or move it.

Open data path (ODP). A control block containing information about the merged file attributes and information returned by I/O operations.

Pop-up window. An area of the screen with visible borders, which supplements the dialog occurring in the full- screen panel or in a previous pop-up window.

Pull-down field choice. A choice that appears in a pull-down menu.

Trimming. An operation performed by removing a list entry from the end of the list opposite from the end where the new entry is added.

Variable buffer. A buffer used to pass dialog variables between the application program and the UIM.

Variable record. A named element of a panel group that identifies the content and layout of a buffer of dialog variables.

Window. An area on the display that is treated as a separate display. Windows have visible boundaries and appear to overlay the display from which they are requested.

Add List Entry (QUIADDLE) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Input	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Option	Input	Char(4)
7	List entry handle	Output	Char(4)
8	Error code	I/O	Char(*)

The Add List Entry (QUIADDLE) API adds one new entry to a list. The new entry is inserted immediately before or immediately after the entry identified by the current entry pointer for the list. The new entry can also be inserted at the beginning or at the end of the list. On return to the application program, the current entry points to the newly inserted entry.

Required Parameter Group

Application handle
INPUT; CHAR(8)
A value assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
INPUT; CHAR(*)
Program storage containing values of one or more dialog variables, from which dialog variable values are copied. Dialog variables are copied in the order specified in the variable record definition.

If the variable record name parameter specifies the name of a variable record defined in the panel group for an open application, dialog variables are copied from the variable buffer to the application variable pool before the list entry is added. This parameter operation is the same as using the Put Dialog Variable (QUIPUTV) API immediately before the QUIADDLE API.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the variable record definition specified in the variable record name parameter. If the buffer is not large enough, an error condition is reported.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied from the variable buffer to the application variable pool. The variable record must be defined in the panel group for the open application. The following special value can be used:

***NONE** The QUIPUTV API is not used during the QUIADDLE API. The variable buffer parameter and variable buffer length are ignored.

List name
INPUT; CHAR(10)
The name of the list to which an entry is added. If the list is not currently active in the open application, it is activated by this API.

Option
INPUT; CHAR(4)
The location of the new entry in the list. When an entry is added to the list, the current entry is always changed to point to the new list entry.

One of the following values must be specified to indicate the new entry's location:

FRST Added as the first entry in the list
LAST Added as the last entry in the list
NEXT Added after the current entry
PREV Added before the current entry

List entry handle
OUTPUT; CHAR(4)
A value representing an entry in a UIM list and returned to the application program representing the current entry in the list until it is removed from the list, even if other entries are inserted and removed from the list. This value is the handle of the entry just inserted in the list.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6AA1 E The value of the action field is not correct at this time. Reason code &5.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A2B E Value for Option parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A36 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A37 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A9D E Size limit reached for list &4.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A93 E Operation not valid when current entry is &5.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Add List Multiple Entries (QUIADDLM) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Input	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Option	Input	Char(4)
7	List entry handle	Output	Char(4)
8	Number of records	Input	Binary(4)
9	Record numbers	Input	Char(*)
10	Record size	Input	Binary(4)
11	Record count	Output	Binary(4)
12	Error code	I/O	Char(*)

The Add List Multiple Entries (QUIADDLM) API adds one or more new entries to a list. The new entries are inserted immediately before or immediately after the entry identified by the current entry for the list. They can also be inserted at the beginning or end of the list. On return to the application program, the current entry points to the most recently inserted entry.

The contents of all dialog variables corresponding to columns in the list are saved in each added entry. When the operation completes successfully, the corresponding dialog variables contain the values from the last list entry successfully added.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
INPUT; CHAR(*)
The program buffer from which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record definition.

If the variable record name parameter specifies the name of a variable record defined in the panel group for an open application, dialog variables are copied from the variable buffer to the application variable pool before the list entry is added. This parameter operation is the same as using the Put Dialog Variable (QUIPUTV) API immediately before the Add List Multiple Entry (QUIADDLM) API.

The variable buffer must be large enough to contain all the variables specified in the variable record definition.

When the number of records parameter has a value greater than one and the record numbers parameter has a value of zero, the size of the variable buffer must be at least equal to the value on the number of records parameter multiplied by the value on the record size parameter.

When the number of records parameter has a value greater than one and the record numbers parameter has a nonzero value, the size of the variable buffer must be at least equal to the largest value in the array of record numbers multiplied by the value of the record size parameter.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the variable record definition specified in the the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record determining which dialog variables are copied from the variable buffer to the application variable pool. The variable record must be defined in the panel group for the open application.

The following special value can be used:

***NONE** The QUIPUTV API is not used during the QUIADDLM API. The variable buffer, variable buffer length, number of records, record size, and record numbers parameters are ignored.

List name
INPUT; CHAR(10)
The name of the list to which entries are added. If the list is not currently active in the open application, it is activated by this API.

Option
INPUT; CHAR(4)
The location of the new entry in the list. When an entry is added to the list, the current entry is always changed to point to the new list entry.

One of the following values must be specified to indicate the new entry's location:

FRST	Added as the first entry in the list
LAST	Added as the last entry in the list
NEXT	Added after the current entry
PREV	Added before the current entry

List entry handle
OUTPUT; CHAR(4)
The list entry handle returned to the application program from the current entry. This value is the handle of the entry just inserted into the list. If more than one list entry is added, this parameter contains the list entry handle of the last entry successfully added.

Number of records
INPUT; BINARY(4)
The total number of entries the application program wants to add to the list. If the variable record name parameter specifies the name of a variable record defined in the panel group for this open application, this parameter can be used with the record size, record count, and record numbers parameters to add multiple entries to the list. When the variable record name parameter has the value *NONE, the number of records parameter is ignored.

The following special value can be used:

1 Only one entry is added to the list. The record size, record count, and record numbers parameters are ignored.

When the number of records parameter is greater than 1, the variable buffer parameter must contain all the entries to be added to the list. The record size parameter defines the size of each record within the variable buffer, and is used to calculate the offset of each record from the first record. The variable record, identified by the variable record name parameter, defines the format of each record.

Record numbers
INPUT; CHAR (*)
An array of record numbers. Each entry in the array is a BINARY(4) value from 1 to 32767. The array specifies the order in which entries are added to the list. The first record number defines the first entry added to the list, the second record number defines the second entry, and so forth. Each record number specifies a record from the variable buffer containing the values for an entry to be added to the list. The dimension of the array must be at least as large as the value for the number of records parameter.

The following special value can be used in the first array element:

0 Records from the variable buffer are used in sequential order. The array of record numbers needs to have only one element.

When the record numbers parameter is 0, the size of the variable buffer must be at least equal to the value specified on the number of records parameter multiplied by the value on the record size parameter. When the record numbers parameter has a nonzero value, the size of the variable buffer must be at least equal to the largest value specified in the record numbers array multiplied by the record size value.

The record numbers parameter allows applications to add more than one list entry from data structures already available to the application program. This is useful when the existing order of records in the data structures is not desired for the list entries, or when certain records should not be added to the list.

Record size
INPUT; BINARY(4)
The size of each record within the variable buffer when multiple records are added to the list. The record size is also used to calculate the offset of each record after the first record.

When the number of records parameter is 1, this parameter is ignored.

Record count
OUTPUT; BINARY(4)
The number of list entries actually added. Validation is done on packed or zoned data values in each record. If an error is found, the Add List Multiple Entries (QUIADDLM) API ends at that point with an exception; unusable data causes a partial update of the list. This parameter returns the number of list entries successfully added to the application program, even if an error occurs.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6AA1 E The value of the action field is not correct at this time. Reason code &5.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.

CPF6A0F E Previous error occurred while running application &3.
CPF6A06 E Record size or record number too large.
CPF6A2B E Value for Option parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A30 E Value for Record Numbers parameter not valid.
CPF6A36 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A37 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A9D E Size limit reached for list &4.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A93 E Operation not valid when current entry is &5.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Add Pop-Up Window (QUIADDPW) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Pop-up window location	Input	Char(10)
3	Row	Input	Binary(4)
4	Column	Input	Binary(4)
5	Error code	I/O	Char(*)

The Add Pop-Up Window (QUIADDPW) API begins the ability to display panels within a pop-up window. All subsequent panel displays for the open application remain in the pop-up window until the Remove Pop-up Window (QUIRMVPW) API is called, or until the QUIADDPW API is called again to add another pop-up window. The maximum number of pop-up windows that can be added to an open application is 20.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Pop-up window location
INPUT; CHAR(10)
The name of a field used for field-adjacent positioning. Field-adjacent positioning places a pop-up window near a field on the underlying panel. The field name must correspond to a name specified on the NAME attribute on one of the following tags of the immediately underlying panel or pop-up window:

- Command Line (CMDLINE)
- Data Item Group (DATAGRP)
- Data Item (DATAI)
- List Column (LISTCOL)
- List Column Group (LISTGRP)
- Menu Item (MENUI)
- Option Line (OPTLINE)

One of these special values may be used to position the window:

***OFFSET** The pop-up window is offset from the top left corner of the most recent underlying panel or pop-up window.

***PULLDOWN** The pop-up window is positioned adjacent to the previous position of the pull-down choice. If the most recent action for the most recently displayed panel was not the selection of a pull-down choice, offset positioning is used.

***ROWCOL** The row and column parameters indicate where the upper left corner of the pop-up window appears. Row and column positioning should be used only when displaying a pop-up window over a non-UIM panel, in conjunction with the Set Screen Image (QUISETSC) API.

Row
INPUT; BINARY(4)
The absolute row used when row and column positioning is requested. The value is either a positive or negative integer.

If a negative integer is provided, the last row for the pop-up window is calculated using the absolute value of the integer, relative to the bottom of the display. A pop-up window cannot begin in row one, and a minimum-depth pop-up window must fit at the specified first row.

When a negative integer is provided, the absolute location of the window is determined by the size of the first panel displayed in the pop-up window. If another panel is displayed in the same window with a smaller width or depth, the location is not recalculated based on the negative integer. If another panel is displayed in the same window with a larger width or depth, the window is changed to use offset positioning.

Column
INPUT; BINARY(4)
The absolute column used when row and column positioning is requested. The value is either a positive or a negative integer.

If a negative integer is provided, the last column for the pop-up window is calculated using the absolute value of the integer, relative to the right side of the display. A pop-up window cannot begin in column one, and a minimum-width pop-up window must fit at the specified first column.

When a negative integer is provided, the absolute location of the window is determined by the size of the first panel displayed in the pop-up window. If another panel is displayed in the same window with a smaller width or depth, the location is not recalculated based on the negative integer. If another panel is displayed in the same window with a larger width or depth, the window is changed to use offset positioning.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3E E Application not open for display.
CPF6A81 E Window cannot be added at this time.
CPF6A82 E &4 is not a valid window location.
CPF6A83 E Row or column given for positioning is not valid.
CPF6A85 E Attempted to display more than &4 windows at a time.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Add Print Application (QUIADDPA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Qualified printer file name	Input	Char(20)
3	Alternative file name	Input	Char(10)
4	Share open data path	Input	Char(1)
5	User data	Input	Char(10)
6	Error code	I/O	Char(*)

Optional Parameter Group:

7	Open data receiver	Output	Char(*)
8	Length of open data receiver	Input	Binary(4)
9	Length of available open data	Output	Binary(4)

The Add Print Application (QUIADDPA) API enables print functions in a previously opened display application by opening the printer file for the application. The QUIADDPA API and the Remove Print Application (QUIRMVPA) API are used in pairs to add and remove printing from applications.

Because the QUIADDPA API requires an open application for display, this print function does not work in a batch environment. For printing in batch, use the Open Print Application (QUIOPNPA) API.

Authorities and Locks

Library Authority *READ
Printer Device File Authority *USE
Printer Device File Lock *SHRNUP

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Qualified printer file name
INPUT; CHAR(20)
The name of the printer device file used for print operations. The first 10 characters contain the name of the *FILE object, and the second 10 characters contain the name of the library in which the printer device file resides. These special values can be used for the library name:

***CURLIB** The job's current library
***LIBL** The library list

The user must have *USE authority to the file named by this parameter.

Alternative file name
INPUT; CHAR(10)
An alternative name for the spooled file. The following special value can be used:

***NONE** There is no alternative name for the spooled file. The name of the spooled file is the name of the printer device file.

Share open data path
INPUT; CHAR(1)
Indicates whether or not the open data path (ODP) for the printer file is shared. Sharing the ODP allows multiple UIM applications to print to the same spooled file. One of the following values must be used:

Y The ODP is shared.
N The ODP is not shared.
F Use the share value of the printer file.

User data
INPUT; CHAR(10)
User data associated with the spooled file. This data becomes an attribute of the spooled file. The following special values can be used:

***FILE** The user data of the spooled file will be set to the user data value of the printer file being opened.

***NONE** There is no user data associated with the spooled file. The user data value of the printer file being opened will be set to blanks.

Error code
I/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

Open data receiver
OUTPUT; CHAR(*)
The variable that is to receive the open data information requested. For the format of the open data receiver variable, see "Format of Data Returned."

Length of open data receiver
INPUT; BINARY(4)
The amount of data the application program is prepared to receive. If the length specified is larger than the amount of data available, the receiver is not changed beyond the amount of data available. If the length specified is larger than the actual length of the open data receiver parameter, unpredictable results may occur.

Length of available open data
OUTPUT; BINARY(4)
The length of all open data available. All available open data is returned if enough space is provided.

Format of Data Returned

The format of the data available, returned in the open data receiver parameter, is as follows:

CHAR(1) Whether or not an error occurred while attempting to obtain the conversion tables needed to process the panel group. The conversion tables are needed when the CHRID attribute of the panel group is not equal to the CHRID attribute of the device, or when the CHRID attribute of the panel group is *JOBCCSID and the job CCSID is not equal to the device CHRID. A CPD6A2A diagnostic message will be logged in the job log for each conversion table that is not found.

One of the following values is returned:

N No error occurred while obtaining the conversion tables or the conversion tables were not necessary.

Y An error occurred while obtaining the conversion tables.

CHAR(1) Whether or not the conversion of data from the process to the device and from the device to the process will result in loss of fidelity of the data. Conversion will be done when the CHRID attribute of the panel group is not equal to the CHRID attribute of the device, or when the CHRID attribute of the panel group is *JOBCCSID and the job CCSID is not equal to the device CHRID.

One of the following values is returned:

N No loss of fidelity will occur.

Y Loss of fidelity may occur on the conversions.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A1A E Application already has an open print file.
CPF6A1C E Unable to add print function.
CPF6A1E E Object cannot be used with this device or print file.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A20 E Print code page not identical to display code page.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3A E Value for Open Data Receiver parameter not valid.
CPF9850 E Override of printer file &1 not allowed.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Close Application (QUICLOA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Close option	Input	Char(1)
3	Error code	I/O	Char(*)

The Close Application (QUICLOA) API closes a UIM application that was opened using the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API. The open and close APIs must be used in pairs to open and close each UIM application.

The QUICLOA API releases all UIM resources associated with the open application, destroying the variable pool and deleting any associated active lists. The storage for internal control blocks supporting the application is freed, and all locks acquired for the open application are released.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The open application for which this API is called. The application handle is assigned by the UIM and returned to the application program when the application is opened.

Close option
INPUT; CHAR(1)
Indicates whether to perform a normal or abnormal close operation. One of the following values must be specified:

A An abnormal close operation is performed. This operation is used when the application prematurely ends processing in the most efficient manner possible. When an application with an open printer file is closed with the abnormal option, the printer trailer text specified on the print trailer message (PRTTRAIL) tag is not printed.

M A normal close operation is performed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Delete List (QUIDLTL) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	List name	Input	Char(10)
3	Error code	I/O	Char(*)

The Delete List (QUIDLTL) API deletes an active list and provides a way for the application to start over with a new list.

QUIDLTL does not need to be used before the Close Application (QUICLOA) API because all lists associated with an open application are automatically deleted when the application is closed.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or Open Print Application (QUIOPNPA) API when the application is opened.

List name
INPUT; CHAR(10)
The name of the list to be deleted. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) API, or its attributes are set with the Set List Attributes (QUISETLA) API.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Display Panel (QUIDSPP) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Function requested	Output	Binary(4)
3	Panel name	Input	Char(10)
4	Redisplay option	Input	Char(1)
5	Error code	I/O	Char(*)

Optional Parameter Group 1:

6	User task	Input	Char(1)
7	Call stack counter	Input	Binary(4)
8	Call message queue	Input	Char(*)
9	Message reference key	Input	Char(4)
10	Cursor position option	Input	Char(1)
11	Last list entry	Input	Char(4)
12	Error list entry	Input	Char(4)
13	Wait time	Input	Binary(4)

Optional Parameter Group 2:

14	Length of call message queue name	Input	Binary(4)
15	Call qualification	Input	Char(20)

The Display Panel (QUIDSPP) API displays a panel and waits for the user to press either a function key or the Enter key.

The values for all I/O fields in the panel definition are taken from dialog variables in the variable pool. If the panel contains list areas, these values are also taken from list entries in the lists associated with the open application.

The application program can also control which list entry is initially displayed at the top of a list area by using the Set List Attributes (QUISETLA) API. If the panel contains a list area displaying an incomplete list and if the list does not have enough entries to fill the list area, the UIM calls the program identified by the program dialog variable parameter of the QUISETLA API to add more list entries. The program is called repeatedly until the requested number of entries is added or until the list is marked as complete.

If the panel contains an extended action list area and if the list is not currently active in the open application, the list is activated by the QUIDSPP API.

Any information the user enters into input fields on the panel is validated according to the specifications of the tag language, then saved in the corresponding dialog variables and list entries.

Control returns to the application program only after all necessary functions are completed as defined by UIM processing rules and tag language specifications. The application program can retrieve dialog variables and list entries after the QUIDSPP API returns to determine what values were specified by the user.

The QUIDSPP API can be called with a long or short argument list. For the short argument list, arguments must be passed to the QUIDSPP API with the required parameters described below. For the long argument list, arguments must be passed to the QUIDSPP API with all the required and optional parameters described below.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Function requested
OUTPUT; BINARY(4)
The function requested by the user. Only special UIM-defined functions and functions using the RETURN dialog command are returned to the application program. Any other functions requested by the user are either handled by the UIM or rejected. If they are rejected, an error message is displayed on the panel.

The RETURN dialog command can be specified as follows:

- On the ACTION attribute of the key list item (KEYI), pull-down field choice (PDFLDC), or menu item (MENUI) UIM tags
- On the ENTER and SELECT attributes of the display panel (PANEL) UIM tag

The RETURN dialog command returns numbers from 1 through 32 767. The return values for UIM-defined functions are as follows:

0 The enter function is requested. This value is returned only for a panel with a list area of ACTOR=CALLER, and only when the user has entered options in the list.

-4 The exit function is requested.

-8 The cancel function is requested.

-10 The prompt function is requested. This value is returned only for a panel with an ACTOR=CALLER list area, and only when the user has entered options in the list.

-20 No function was requested before the time specified on the wait time parameter has ended. This value is returned only when the wait time parameter specifies a time-out value.

Panel name
INPUT; CHAR(10)
The name of the panel to display. The panel must be defined in the panel group for the open application.

Redisplay option
INPUT; CHAR(1)
Indicates whether or not the panel is formatted and displayed as a first-time display or as a redisplay.

One of the following values must be used:

Y The panel is formatted as a redisplay. If the panel has not been displayed in the application before, processing for the first-time display is done and this parameter is ignored.

Cursor positioning that is controlled by the program is ignored when the redisplay option is used. Cursor positioning controlled by action list processing is supported when the redisplay option is used.

Dialog variables are edited for a new displayable value only if the variable pool contains a value more current than the value from the last time the panel was displayed. When VARUPD=NO is processed, the panel is redisplayed with values entered in input fields but not stored in the variable pool.

N The panel is formatted as a first-time display. The panel is displayed with the first conditioned-on item at the top of each pageable data and menu area. Pageable information areas are positioned at the first line of text. All dialog variables displayed on the screen are edited to produce a displayable value.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Optional Parameter Group 1

User task
INPUT; CHAR(1)
Indicates whether or not this panel is the logical start of a new user task. The value used determines whether or not the UIM returns to the application program when the EXIT dialog command is requested by a lower level program, such as an action on the KEYI tag.

The user task has no effect on UIM operations when the EXIT dialog command is specified directly in the definition of this panel as the function key, pull-down choice, or menu item action.

One of the following values must be specified:

N This panel is the logical start of a new user task. If the user generates the EXIT dialog command at a lower call level (for example, by pressing the exit function key on a panel displayed by a list action from the original panel), this panel is redisplayed without returning control to the application program.

O This panel is not the start of a new user task. If the user generates the EXIT dialog command at a lower call level, control returns to the application program with an indication that the exit function was requested. This is the default value when a short argument list is passed to the QUIDSPP API.

Call stack counter
INPUT; BINARY(4)
A number identifying the location in the program stack of the call message queue for the UIM to use. This parameter is used with the call message queue parameter.

Any nonnegative number can be specified for the relative program call number.

0 The message queue of the program specified in the call message queue parameter. This is the default value when a short argument list is passed to the QUIDSPP API.

1 The message queue of the program that calls the program specified in the call message queue parameter is used for messages displayed to the user.

n The message queue of the *n*th program up the stack from the program specified in the call message queue parameter is used for messages displayed to the user.

Call message queue
INPUT; CHAR(*)
The name of the message queue for the UIM to use, or the name of the program to start counting from when using a value other than zero for the call stack counter parameter. The program you specify must be in the call stack. The UIM will:

- Receive a message from this queue to initially display on the message line of the panel. This is done using the value passed for the message reference key parameter.
- Move to this message queue all completion, information, diagnostic, and escape messages sent to the UIM by programs or commands called to process an action defined in the panel. Note that escape messages are changed to diagnostic messages when they are moved.

The following special value can be used:

***CALLER** The application program calling the QUIDSPP API is the starting point for identifying the call message queue. This is the default value when a short argument list is passed to the QUIDSPP API.

This parameter can be from 1 to 256 characters in length. The length of call message queue parameter specifies this parameter's length. If the length of call message queue parameter is not used, then the length of this parameter is assumed to be 10 charac-

ters. If further qualification is needed to identify the call, the call qualification parameter can be used.

Message reference key
INPUT; CHAR(4)
The messages in the call message queue that are displayed on the panel. The call message queue is identified by the call stack counter parameter and the call message queue parameter. This parameter must be set to the message reference key for the first (oldest) message on the call message queue that should be displayed on the panel. One of the following special values can be used:

(blank) No messages are shown to the user on the initial panel, even if there are messages in the call message queue. This is the default value when a short argument list is passed to the Display Panel (QUIDSPP) API.

FRST All new messages in the call message queue, starting with the first new message in the queue, are presented on the initial panel.

Cursor position option
INPUT; CHAR(1)
The rules that determine the initial position of the cursor when the panel is displayed. This parameter controls only the initial cursor placement when the panel is displayed by the QUIDSPP API. When a panel is redisplayed by the UIM or by the application program, the cursor remains where it was left by the user.

One of the following values must be used:

A Cursor positioning for action list processing is performed. This value is used when the panel is redisplayed after action list processing when ACTOR=CALLER is on the list area (LIST) tag. The cursor and list are positioned as defined by the last list entry and error list entry parameters.

D Default cursor positioning is performed. The specific cursor position depends on whether or not any variables or list entries associated with the panel are marked in error. This is the default value when a short argument list is passed to the display panel (QUIDSPP) API.

P Cursor positioning that is controlled by the program is performed. The cursor is positioned using the current value of the dialog variables associated with the CSRVAR, CSRPOS, CSRLST, and CSREID attributes of the panel definition (PANEL) tag. Cursor positioning controlled by the program is allowed only for a panel containing all four cursor positioning attributes.

With cursor positioning controlled by the program, each pageable, nonlist area is repositioned to make sure that the first input field in error is visible, but the cursor is not positioned at the first input field in error. If the value of any of the dialog variables for cursor position identified in the panel definition is unusable, the cursor defaults to the first input field on the panel.

Last list entry
INPUT; CHAR(4)
The list entry handle for the last action list entry that had an action processed.

This information is used to position the cursor. Cursor positioning for action list processing must be specified in the cursor position option parameter or this parameter is ignored. One of the following special values can be used:

EXTE The last list action processed is for the extended action entry.

NONE The application is providing no value for this parameter. This is the default value when a short argument list is passed to the Display Panel (QUIDSPP) API.

Error list entry
INPUT; CHAR(4)
The list entry handle for the first and only list entry that had an error while processing list actions. This parameter is also set when action list processing is stopped because a user pressed F3 (Exit) or F12 (Cancel). This information displays the correct page of list information.

Cursor positioning for action list processing must be specified in the cursor position option parameter, or this parameter is ignored.

One of the following special values can be used:

EXTE The extended action entry is in error; the UIM does not reposition the list.

NONE The application is providing no value for this parameter. This is the default value when a short argument list is passed to the Display Panel (QUIDSPP) API.

Wait time
INPUT; BINARY(4)
The number of seconds to wait for data to become available from the work station. After this amount of time, the keyboard locks, control returns to the application, and the function requested parameter returns an indication that the wait time ended. Because the wait time ended, input dialog variables associated with the panel are not updated. The number of seconds specified should be a positive integer.

One of the following special values can be used:

-1 The UIM waits indefinitely until data becomes available from the work station. This is the default value when a short argument list is passed to the Display Panel (QUIDSPP) API.

0 The UIM does not wait for data to become available from the work station. The panel is displayed with the keyboard locked, and control returns immediately to the application with an indication in the function requested parameter that the wait time ended.

Optional Parameter Group 2

Length of call message queue name
INPUT; BINARY(4)
The length of the call message queue name. Valid values for this parameter range from 1 to 256. If the value is not valid, an error will occur.

The default value for this parameter is 10.

Call qualification
INPUT; CHAR(20)
The name of the module and bound program that contain the procedure. This value is used to further qualify the procedure identified in the call message queue parameter. The first 10 characters specify the module name, and the second 10 characters specify the bound program name. When the call qualification parameter is specified, the call stack will be searched only for a procedure within a bound program.

If this parameter is not used, only the call message queue parameter will be used to identify the call.

The special value of *NONE can be used for the module or bound program name, or for both. When *NONE is specified for one of the names, then that name will not be used when searching for the qualified procedure. If *NONE is specified for both names, only the call message queue parameter will be used to identify the call. Original program model (OPM) programs should specify *NONE for both names.

Error Messages

CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A13 E Application &3 closed prematurely.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A22 E Panel cannot be displayed in a window.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3E E Application not open for display.
CPF6A3F E &4 was not found in panel group &1.
CPF6A4A E &4 is too large to show as a window.
CPF6A4B E Value for Redisplay parameter not valid.
CPF6A40 E &4 is already in use.
CPF6A41 E CSRPOS(*DFT) required for display &4.
CPF6A42 E Display of panel &4 not allowed.
CPF6A43 E Cannot use program message queue &6.
CPF6A45 E Module or bound program name not valid.
CPF6A50 E Error was found during display file or print file operation.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Get Dialog Variable (QUIGETV) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Output	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	Error code	I/O	Char(*)

The Get Dialog Variable (QUIGETV) API allows a program to obtain the value of one or more dialog variables by specifying the application program's variable buffer and the name of a variable record defined in the panel group for the open application.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
OUTPUT; CHAR(*)
The program buffer into which dialog variable values are copied. The variable buffer must be large enough to contain all the variables specified in the definition of the variable record. The dialog variables are copied in the order specified in the variable record, which is specified in the variable record name parameter.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the definition of the variable record, which is specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record definition that determines which dialog variables are copied from the application variable pool to the variable buffer. The variable record must be defined in the panel group for the open application.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Get List Entry (QUIGETLE) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Output	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Positioning option	Input	Char(4)
7	Copy option	Input	Char(1)
8	Selection criteria	Input	Char(20)
9	Selection handle	Input	Char(4)
10	Extend option	Input	Char(1)
11	List entry handle	Output	Char(4)
12	Error code	I/O	Char(*)

The Get List Entry (QUIGETLE) API accesses an entry in a list and optionally updates the corresponding dialog variables to the values in the list entry.

The entry accessed depends on the positioning option parameter and the value of the current entry pointer for the list. On return to the application program, the current entry pointer in the list points to the entry retrieved, except when retrieving the extended action entry. The extended action entry is a pseudo list entry that can be retrieved using this API, and can be updated using the Update List Entry (QUIUPDLE) API.

If requested, the corresponding dialog variables contain the values from the list entry retrieved.

If an error variable corresponding to a field in the list entry is specified in the variable record, identified by the variable record name parameter, it is set according to the error state of the corresponding dialog variable.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) or Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
OUTPUT; CHAR(*)
The program buffer into which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record definition.

If the variable record name parameter specifies the name of a variable record defined in the panel group for this open application, dialog variables are copied into the variable buffer from the application variable pool after the list entry is retrieved. This parameter does the same job as using the Get Dialog Variable (QUIGETV) API immediately after the QUIGETLE API.

The variable buffer must be large enough to contain all the variables specified in the variable record definition.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the variable record definition, which is specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied between the application variable pool and the variable buffer. The variable record must be defined in the panel group for the open application. The following special value can be used:

***NONE** The QUIGETV API is not done during the QUIGETLE API; the parameter for the variable buffer is ignored when this value is used.

List name
INPUT; CHAR(10)
The name of the list from which an entry is retrieved. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) API or Add List Multiple Entries (QUIADDLM) API, or when the list attributes are set with the Set List Attributes (QUISETLA) API.

Positioning option
INPUT; CHAR(4)
The placement of the current entry pointer to the list entry specified. If a positioning error occurs, the current list position is not changed. One of the following values must be specified to set the entry pointer to the appropriate position:

Value	Pointer Position
BOT	The bottom of the list. A special position just after the last list entry in a list that is complete at the bottom.
FRST	The first entry in the currently built list. If the list is empty, an error is reported.
FSLT	The entire list is searched in a forward direction to locate an entry satisfying the condition specified by the selection criteria parameter. If the list is empty, an error is reported. The current entry pointer for the list entry is repositioned at the first entry in the list and the search proceeds forward, including the first entry. The search does not wrap.
HNDL	The list entry specified by the list entry handle parameter.
LAST	The last entry in the currently built list. If the list is empty, an error is reported.
LSLT	The entire list is searched in a backward direction to locate an entry satisfying the condition specified by the selection criteria parameter. If the list is empty, an error is reported. The current list entry pointer is repositioned to the last entry in the list and the search proceeds backward, including the last entry. The search does not wrap.
NEXT	The current entry pointer is advanced one entry, pointing to the next entry in the list.
NSLT	The list is searched in a forward direction to locate an entry satisfying the condition specified by the selection criteria parameter. The search starts with the entry after the current list entry pointer and does not wrap.
PREV	The current entry pointer is moved back one entry, pointing to the previous entry in the list.
PSLT	The list is searched in a backward direction to locate an entry satisfying the condition specified by the selection criteria parameter. The search starts with the entry before the current list entry pointer and does not wrap.
SAME	The current entry pointer remains unchanged, pointing to the same entry in the list.
TOP	The top of the list. A special position just prior to the first list entry in a list that is complete at the top.

Note: After running the QUIGETLE API once with the positioning option parameter of FSLT or LSLT, the application program should change the positioning option parameter to NSLT or PSLT. It does this to avoid repeatedly positioning at the beginning or end of the list and searching the same list entries.

Copy option

INPUT; CHAR(1)

Determines whether or not the data values in the list entry are copied into the corresponding dialog variables when positioning is complete. One of the following values must be specified:

Y The data values in the current list entry after the positioning operations are performed are copied into corresponding dialog variables. This value must be specified if the variable record name parameter specifies the name of a variable record defined in the panel group for the open application. This value is not allowed when the current entry is positioned to TOP or BOT.

N The data values in the current list entry after positioning the list are not copied into corresponding dialog variables.

Selection criteria

INPUT; CHAR(20)

The selection criteria used when positioning the list by variable selection. Positioning by variable selection allows the application program to search forward to find the next entry, or backward to find a previous entry satisfying a given condition. The search is not dependent on sorted list entries, but the application program might need to build the list in sorted order to get a consistent result when application comparison operators are other than equal (EQ) or not equal (NE).

This parameter is only used when one of the following values is specified for the positioning option parameter: FSLT, LSLT, NSLT, or PSLT. It is ignored if any other value is specified.

The relational condition defined by this parameter must exist between the value of a dialog variable and that dialog variable's corresponding value in a list entry when selection positioning is performed.

The first 10 characters of this parameter must contain a comparison operator. The operator must be left-adjusted and padded with blanks.

One of the following values must be specified for the comparison operator:

EQ The list entry value equals the dialog variable value.

NE The list entry value is not equal to the dialog variable value.

GT The list entry value is greater than the dialog variable value.

LT The list entry value is less than the dialog variable value.

GE The list entry value is greater than or equal to the dialog variable value.

LE The list entry value is less than or equal to the dialog variable value.

The second 10 characters of this parameter must contain the name of the dialog variable used in the comparison. The dialog variable name specified must be defined in the list being searched.

Selection handle
INPUT; CHAR(4)
The list entry handle used when positioning the current entry pointer to a specific entry. This parameter is used only when the positioning option parameter has the value HNDL; it is ignored if any other value is specified.

The following special value can be specified:

EXTE Retrieves the contents of the extended action entry. The contents of the extended action entry are copied to the associated dialog variables in the variable pool. However, the current entry pointer for the list is not changed when the extended action entry is retrieved.

A list entry handle uniquely distinguishes an entry until it is removed from the list, even if other entries are inserted and removed from the list.

Extend option
INPUT; CHAR(1)
Indicates whether or not an incomplete list is automatically extended in the attempt to retrieve the requested list entry. The list can be extended when one of the following values is specified for the positioning option parameter: NEXT, PREV, TOP, BOT, NSLT, PSLT, FSLT, or LSLT. This parameter is ignored and the list is not extended if any other value is specified for the positioning option parameter.

One of the following values must be specified:

Y The list is extended, if necessary, to find the requested list entry.

N The list is not extended to find the requested list entry. This option can be used if the program calling the QUIGETLE API needs to process only entries already in the list.

List entry handle
OUTPUT; CHAR(4)
The list entry handle from the current entry pointer in the list. This value is the handle of the entry to which the list was last positioned by this API. The following special values may be returned:

TOP The current entry pointer is positioned at the top of the list.
BOT The current entry pointer is set at the bottom of the list.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A2C E Value for Option parameter not valid.
CPF6A2D E Value for Selection Criteria parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A27 E Value for Extend Option parameter not valid.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF6A93 E Operation not valid when current entry is &5.
CPF6A95 E List &4 either not complete or not extended.
CPF6A96 E Variable &5 is not in list definition.
CPF6A97 E Variable &5 not valid type for select comparison.
CPF6A98 E Entry not found in list &4 in panel group &1 in &2.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

Get List Multiple Entries (QUIGETLM) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Output	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Positioning option	Input	Char(4)
7	Copy option	Input	Char(1)
8	Selection criteria	Input	Char(20)
9	Selection handle	Input	Char(4)
10	Extend option	Input	Char(1)
11	List entry handle	Output	Char(4)
12	Number of records	Input	Binary(4)
13	Record size	Input	Binary(4)
14	Record count	Output	Binary(4)
15	Error code	I/O	Char(*)

The Get List Multiple Entries (QUIGETLM) API accesses one or more entries in a list and updates the corresponding dialog variables with the values contained in the list entry.

The entry accessed depends on the positioning option parameter and the value of the current entry pointer for the list. On return to the application program, the current entry pointer in the list points to the last entry retrieved, except when retrieving the extended action entry. The extended

action entry can be retrieved using this API, and can be updated using the Update List Entry (QUIUPDLE) API.

When the operation completes successfully, the corresponding dialog variables contain the values from the last list entry retrieved.

If an error variable corresponding to a field in the list entry is specified in the variable record, which is identified by the variable record name parameter, it is set according to the error state of the corresponding dialog variable.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
OUTPUT; CHAR(*)
The program buffer into which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record definition.

Dialog variables are copied into the variable buffer from the application variable pool after the list entry is retrieved. This parameter does the same job as using the Get Dialog Variable (QUIGETV) API immediately after the list entry is retrieved.

The variable buffer must be large enough to contain all the variables specified in the variable record definition.

When the number of records parameter is greater than 1, the size of the variable buffer must be at least equal to the value specified on the number of records parameter multiplied by the value specified on the record size parameter.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the variable record definition specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied between the application variable pool and the variable buffer. The variable record must be defined in the panel group for the open application.

The special value of *NONE may not be used with the QUIGETLM API.

List name
INPUT; CHAR(10)
The name of the list from which an entry is retrieved. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) or Add List Multiple Entries (QUIADDLM) API, or when the list attributes are set with the Set List Attributes (QUISETLA) API.

Positioning option
INPUT; CHAR(4)
Sets the current entry pointer to the list entry specified. If a positioning error occurs, the current list position is not changed. One of the following values must be specified to set the entry pointer to the appropriate position:

Value	Pointer Position
BOT	The bottom of the list. A special position just after the last list entry in a list that is complete at the bottom.
FRST	The first entry in the currently built list. If the list is empty, an error is reported.
FSLT	The entire list is searched in a forward direction to locate an entry satisfying the condition specified by the selection criteria parameter. If the list is empty, an error is reported. The current pointer for the list entry is repositioned at the first entry in the list, and the search proceeds forward, including the first entry. The search does not wrap.
HNDL	The list entry specified by the list entry handle parameter.
LAST	The last entry in the currently built list. If the list is empty, an error is reported.
LSLT	The entire list is searched in a backward direction to locate an entry satisfying the condition specified by the selection criteria parameter. If the list is empty, an error is reported. The current list entry pointer is repositioned to the last entry in the list, and the search proceeds backward, including the last entry. The search does not wrap.
NEXT	The current entry pointer is advanced one entry, pointing to the next entry in the list.
NSLT	The list is searched in a forward direction to locate an entry satisfying the condition specified by the selection criteria parameter. The search starts with the entry after the current list entry pointer, and does not wrap.
PREV	The current entry pointer is moved back one entry, pointing to the previous entry in the list.
PSLT	The list is searched in a backward direction to locate an entry satisfying the condition specified by the selection criteria parameter. The search starts with the entry before the current list entry pointer and does not wrap.

SAME The current entry pointer remains unchanged, pointing to the same entry in the list.

TOP The top of the list. A special position just prior to the first list entry in a list that is complete at the top.

Note: After running the QUIGETLM API once with the positioning option parameter of FSLT or LSLT, the application program should change the positioning option parameter to NSLT or PSLT. It does this to avoid repeatedly positioning at the beginning or end of the list and searching the same list entries.

Copy option
INPUT; CHAR(1)
Determines whether or not the data values in the list entry are copied into the corresponding dialog variables when positioning is complete. One of the following values must be specified:

Y The data values in the current list entry after the positioning operations are performed are copied into corresponding dialog variables. This value must be specified if the variable record name parameter specifies the name of a variable record defined in the panel group for the open application. Y must be used if the number of records parameter is greater than 1. This value is not allowed when the current entry is positioned to TOP or BOT.

N The data values in the current list entry after positioning the list are not copied into corresponding dialog variables.

Selection criteria
INPUT; CHAR(20)
The selection criteria used when positioning the list by variable selection. Positioning by variable selection allows the application program to search forward to find the next entry, or backward to find a previous entry satisfying a given condition. The search is not dependent on sorted list entries, but the application program might need to build the list in sorted order to get a consistent result when using comparison operators other than equal (EQ) or not equal (NE).

This parameter is only used when FSLT, LSLT, NSLT, or PSLT is specified for the positioning option parameter. It is ignored if any other value is specified.

The relational condition defined by this parameter must exist between the dialog variable value and that dialog variable's corresponding value in a list entry when selection positioning is performed.

The first 10 characters of this parameter must contain a comparison operator. The operator must be left-adjusted and padded with blanks.

One of the following values must be specified for the comparison operator:

EQ The list entry value equals the dialog variable value.

NE The list entry value is not equal to the dialog variable value.

GT The list entry value is greater than the dialog variable value.

LT The list entry value is less than the dialog variable value.

GE The list entry value is greater than or equal to the dialog variable value.

LE The list entry value is less than or equal to the dialog variable value.

The second 10 characters of this parameter contain the name of the dialog variable used in the comparison. The dialog variable name specified must be defined in the list being searched.

Selection handle
INPUT; CHAR(4)
The list entry handle used when positioning the current entry pointer to a specific entry. This parameter is used only when the positioning option parameter has the value HNDL; it is ignored if any other value is specified.

The following special value can be specified:

EXTE Retrieves the content of the extended action entry. The contents of the extended action entry are copied to the associated dialog variables in the variable pool. However, the current entry pointer for the list is not changed when the extended action entry is retrieved.

A list entry handle uniquely distinguishes an entry until it is removed from the list, even if other entries are inserted and removed from the list.

Extend option
INPUT; CHAR(1)
Indicates whether or not an incomplete list is automatically extended in the attempt to retrieve the requested list entry. The list can be extended when one of the following values is specified for the positioning option parameter: NEXT, PREV, TOP, BOT, NSLT, PSLT, FSLT, or LSLT. If any other value is specified for the positioning option parameter, this parameter is ignored and the list is not extended.

One of the following values must be specified:

Y The list is extended, if necessary, to find the requested list entry.

N The list is not extended to find the requested list entry. This option can be used if the program calling the QUIGETLE API needs to process only entries already in the list.

List entry handle
OUTPUT; CHAR(4)
The list entry handle from the current entry pointer in the

list. This value is the handle of the entry to which the list was last positioned by this API.

The following special values may be returned.

TOP The current entry pointer is positioned at the top of the list.

BOT The current entry pointer is set at the bottom of the list.

Number of records

INPUT; BINARY(4)

The total number of entries to retrieve from the list. The following special value can be used:

1 Only one entry is retrieved from the list. The record size and record count parameters are ignored when this value is used.

When this parameter is greater than 1, the variable buffer must contain space for all the entries retrieved from the list.

When this parameter is greater than 1 (multiple entries are being retrieved), the positioning option parameter must have one of the following values: NEXT, PREV, NSLT, PSLT, FSLT, or LSLT.

Record size

INPUT; BINARY(4)

The size of each record within the variable buffer when multiple records are retrieved from the list. This parameter also calculates the offset of each record after the first one.

When the number of records parameter is 1, this parameter is ignored.

Record count

OUTPUT; BINARY(4)

The number of list entries actually retrieved. When the number of records parameter is 1, this parameter is ignored.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A06 E Record size or record number too large.
CPF6A13 E Application &3 closed prematurely.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A2C E Value for Option parameter not valid.
CPF6A2D E Value for Selection Criteria parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A27 E Value for Extend Option parameter not valid.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF6A93 E Operation not valid when current entry is &5.
CPF6A95 E List &4 either not complete or not extended.
CPF6A96 E Variable &5 is not in list definition.
CPF6A97 E Variable &5 not valid type for select comparison.
CPF6A98 E Entry not found in list &4 in panel group &1 in &2.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Open Display Application (QUIOPNDA) API

Parameters

Required Parameter Group:

1	Application handle	Output	Char(8)
2	Qualified panel group name	Input	Char(20)
3	Application scope	Input	Binary(4)
4	Exit parameter interface	Input	Binary(4)
5	Full-screen help	Input	Char(1)
6	Error code	I/O	Char(*)

Optional Parameter Group:

7	Open data receiver	Output	Char(*)
8	Length of open data receiver	Input	Binary(4)
9	Length of available open data	Output	Binary(4)

The Open Display Application (QUIOPNDA) API initiates a UIM display application by opening the panel group that the application program specifies. The QUIOPNDA API and the Close Application (QUICLOA) API must be used in pairs to open and close each UIM display application.

Multiple applications can be opened at the same time. Each open application contains a complete set of dialog variables and active lists, and is independent of other open applications. A panel group can be opened more than once per job, but each call of the QUIOPNDA API initiates a new UIM display application and returns a unique application handle.

Authorities and Locks

Library Authority *READ
Panel Group Authority *USE
Panel Group Lock *SHRRD

Required Parameter Group

Application handle
OUTPUT; CHAR(8)
The application handle for the opened application. The QUIOPNDA API returns a unique handle for each application opened. This handle must be provided as a parameter to every other UIM API that operates on the application, including the QUICLOA API, which must be used to close the application.

Qualified panel group name
INPUT; CHAR(20)
The name of the *PNLGRP object opened for this UIM application. The first 10 characters contain the name of the *PNLGRP object, and the second 10 characters contain the name of the library in which the panel group resides. These special values can be used for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Application scope
INPUT; BINARY(4)
The scope of the resources for the application. The UIM uses the scope to determine whether or not to automatically close the application when reclaim resource processing is performed through the Reclaim Resource command (RCLRSC), the Reclaim Activation Group command (RCLACTGRP), or the End Request command (ENDRQS). During reclaim resource processing, the UIM closes all applications whose scope is no longer active.

One of the following values must be used:

-1 The calling program is the scope for the application. If the calling program is an original program model (OPM) program and the application program is no longer active, the UIM will automatically close the application during reclaim resource processing. If the calling program is an Integrated Language Environment (ILE) program and the activation group mark is no longer active, the UIM will automatically close the application during reclaim resource processing.

0 The job is the scope for the application. The application is not automatically closed by the UIM until the job is ended.

Exit parameter interface
INPUT; BINARY(4)
The type of parameter interface used with exit programs and programs called as a result of the CALL dialog command for the UIM application being opened.

All exit and CALL programs receive a single parameter or multiple parameters, which are space pointers to information describing the state of the UIM application.

One of the following values must be used:

0 Used by programs written in languages that can efficiently process structures.

1 Used by programs written in languages that cannot efficiently process structures. This value indicates that all application programs called as exits or as a result of the CALL dialog command accept the parameter lists described for interface level 1.

2 Used by programs written in languages that cannot efficiently process structures. This value indicates that all application programs called as exits or as a result of the CALL dialog command accept the parameter lists described for interface level 2.

For a detailed description of the structure passed for each type of exit and CALL program and for a description of the exit parameter lists, see Chapter 59, "User Interface Management Exit Programs" on page 59-1.

Full-screen help
INPUT; CHAR(1)
Determines whether or not the UIM help for the application is displayed in pop-up windows or with a full screen. One of the following values must be used:

Y The online help information is displayed with a full screen.

N The online help information is displayed using pop-up windows, unless the user profile indicates that help is displayed with a full screen.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Optional Parameter Group

Open data receiver
OUTPUT; CHAR(*)
The receiver variable that is to receive the open data information requested. For the format of the open data receiver variable, see "Format of Data Returned" on page 58-20.

Length of open data receiver
INPUT; BINARY(4)
The amount of data the application program is prepared to receive. If the length specified is larger than the amount of data available, the receiver is not changed beyond the amount of data available. If the length specified is greater than the actual length of the open data receiver parameter, unpredictable results may occur.

Length of available open data
OUTPUT; BINARY(4)
The length of all open data available. All available open data is returned if enough space is provided.

Format of Data Returned

The format of the data available, returned in the open data receiver parameter, is as follows:

CHAR(1) Whether or not an error occurred while attempting to obtain the conversion tables needed to process the panel group. The conversion tables are needed when the CHRID attribute of the panel group is not equal to the CHRID attribute of the device, or when the CHRID attribute of the panel group is *JOBCCSID and the job CCSID is not equal to the device CHRID. A CPD6A2A diagnostic message will be logged in the job log for each conversion table that is not found.

One of the following values is returned:

- **N** No error occurred while obtaining the conversion tables or the conversion tables were not necessary.
- **Y** An error occurred while obtaining the conversion tables.

CHAR(1) Whether or not the conversion of data from the process to the device and from the device to the process will result in loss of fidelity of the data. Conversion will be done when the CHRID attribute of the panel group is not equal to the CHRID attribute of the device, or when the CHRID attribute of the panel group is *JOBCCSID and the job CCSID is not equal to the device CHRID.

One of the following values is returned:

- **N** No loss of fidelity will occur.
- **Y** Loss of fidelity may occur on the conversions.

Error Messages

CPF6A1E E Object cannot be used with this device or print file.
CPF6A12 E Unable to open panel group.
CPF6A17 E Panel group &1 in library &2 is not at the current release level.
CPF6A2A E Value for Application Scope parameter not valid.
CPF6A2E E Value for Exit Program Interface parameter not valid.
CPF6A2F E Value for Full Screen Help parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A26 E Resources not available to open application.
CPF6A3A E Value for Open Data Receiver parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Open Print Application (QUIOPNPA) API

Parameters

Required Parameter Group:

1	Application handle	Output	Char(8)
2	Qualified panel group name	Input	Char(20)
3	Application scope	Input	Binary(4)
4	Exit parameter interface	Input	Binary(4)
5	Qualified name of printer device file	Input	Char(20)
6	Alternative file name	Input	Char(10)
7	Share open data path	Input	Char(1)
8	User data	Input	Char(10)
9	Error code	I/O	Char(*)

Optional Parameter Group:

10	Open data receiver	Output	Char(*)
11	Length of open data receiver	Input	Binary(4)
12	Length of available open data	Output	Binary(4)

The Open Print Application (QUIOPNPA) API initiates a UIM print application by opening the panel group that the application program specifies. The QUIOPNPA API and the Close Application (QUICLOA) API must be used in pairs to open and close each UIM print application.

Multiple applications can be opened at the same time. Each open application contains a complete set of dialog variables and active lists, and is independent of other open applications. The same panel group can be opened more than once per job, but each call of the QUIOPNPA API initiates a new UIM print application and returns a unique application handle.

Authorities and Locks

Library Authority	*READ
Panel Group Authority	*USE
Panel Group Lock	*SHRRD
Printer Device File Authority	*USE
Printer Device File Lock	*SHRNUP

Required Parameter Group

Application handle
OUTPUT; CHAR(8)
The application handle for the open application. The Open Display Application (QUIOPNDA) API returns a unique handle for each application opened. This handle must be provided as a parameter to every other UIM API that operates on the application, including the QUICLOA API, which must be used to close the application.

Qualified panel group name
INPUT; CHAR(20)
The name of the *PNLGRP object opened for this UIM

application. The first 10 characters contain the name of the *PNLGRP object, and the second 10 characters contain the name of the library in which the panel group resides. These special values can be used for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Application scope
INPUT; BINARY(4)
The scope of the resources for the application. The UIM uses the scope to determine whether or not to automatically close the application when reclaim resource processing is performed through the Reclaim Resource (RCLRSC) command, the Reclaim Activation Group (RCLACTGRP) command, or the End Request (ENDRQS) command. During reclaim resource processing, the UIM closes all applications whose scope is no longer active.

One of the following values must be used:

-1 The calling program is the scope for the application. If the calling program is an original program model (OPM) program and the application program is no longer active, the UIM will automatically close the application during reclaim resource processing. If the calling program is an Integrated Language Environment (ILE) program and the activation group mark is no longer active, the UIM will automatically close the application during reclaim resource processing.

0 The job is the scope for the application. The application is not automatically closed by the UIM until the job is ended.

Exit parameter interface
INPUT; BINARY(4)
The type of parameter interface used with exit programs and programs called as a result of the CALL dialog command for the UIM application being opened.

All exit and CALL programs receive a single parameter or multiple parameters, which are space pointers to information describing the state of the UIM application.

One of the following values must be used:

0 Used by programs written in languages that can efficiently process structures.

1 Used by programs written in languages that cannot efficiently process structures. This value indicates that all application programs called as exits or as a result of the CALL dialog command accept the parameter lists described for interface level 1.

2 Used by programs written in languages that cannot efficiently process structures. This value indicates that all application programs called as exits or as a result of the CALL dialog command accept the parameter lists described for interface level 2.

For a detailed description of the exit parameter lists, and for a description of the type of structure passed for each type of exit and CALL program, see Chapter 59, "User Interface Management Exit Programs" on page 59-1.

Qualified name of printer device file
INPUT; CHAR(20)
The name of the printer device file used in print operations. The first 10 characters contain the *FILE object name, and the second 10 characters contain the name of the library in which the printer device file resides. These special values can be used for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Alternative file name
INPUT; CHAR(10)
An alternative name for the spooled file. The following special value can be used:

***NONE** There is no alternative name for the spooled file. The name of the spooled file is the name of the printer device file.

Share open data path
INPUT; CHAR(1)
Determines whether or not the printer device file's open data path (ODP) is shared. Sharing the ODP allows multiple UIM applications to print to the same spooled file. One of the following values must be used:

Y The ODP is shared.
N The ODP is not shared.
F Use the share value of the print file.

User data
INPUT; CHAR(10)
User data associated with the spooled file. This data becomes an attribute of the spooled file. The following special values can be used:

***FILE** The user data of the spooled file will be set to the user data value of the printer file being opened.
***NONE** There is no user data associated with the spooled file. The user data value of the printer file being opened will be set to blanks.

Error code
I/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

Open data receiver
OUTPUT; CHAR(*)
The receiver variable that is to receive the open data

information requested. For the format of the open data receiver variable, see "Format of Data Returned" on page 58-22.

Length of open data receiver
INPUT; BINARY(4)
The amount of data the application program is prepared to receive. If the length specified is larger than the amount of data available, the receiver is not changed beyond the amount of data available. If the length specified is greater than the actual length of the open data receiver parameter, unpredictable results may occur.

Length of available open data
OUTPUT; BINARY(4)
The length of all open data available. All available open data is returned if enough space is provided.

Format of Data Returned

The format of the data available, returned in the open data receiver parameter, is as follows:

CHAR(1) Whether or not an error occurred while attempting to obtain the conversion tables needed to process the panel group. The conversion tables are needed when the CHRID attribute of the panel group is not equal to the CHRID attribute of the device, or when the CHRID attribute of the panel group is *JOBCCSID and the job CCSID is not equal to the device CHRID. A CPD6A2A diagnostic message will be logged in the job log for each conversion table that is not found.

One of the following values is returned:

N No error occurred while obtaining the conversion tables or the conversion tables were not necessary.

Y An error occurred while obtaining the conversion tables.

CHAR(1) Whether or not the conversion of data from the process to the device and from the device to the process will result in loss of fidelity of the data. Conversion will be done when the CHRID attribute of the panel group is not equal to the CHRID attribute of the device, or when the CHRID attribute of the panel group is *JOBCCSID and the job CCSID is not equal to the device CHRID.

One of the following values is returned:

N No loss of fidelity will occur.

Y Loss of fidelity may occur on the conversions.

Error Messages

CPF6A1C E Unable to add print function.
CPF6A1E E Object cannot be used with this device or print file.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A12 E Unable to open panel group.
CPF6A17 E Panel group &1 in library &2 is not at the current release level.
CPF6A2A E Value for Application Scope parameter not valid.
CPF6A2E E Value for Exit Program Interface parameter not valid.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A26 E Resources not available to open application.
CPF6A3A E Value for Open Data Receiver parameter not valid.
CPF9850 E Override of printer file &1 not allowed.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Print Panel (QUIPRTP) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Print panel name	Input	Char(10)
3	Eject option	Input	Char(1)
4	Error code	I/O	Char(*)

The Print Panel (QUIPRTP) API prints a panel to the printer file for an opened print application. The values for all output fields used in the panel definition are taken from dialog variables in the variable pool. If the panel contains list areas, the values are also taken from list entries in the lists associated with the open application.

If the panel contains a list area that is incomplete at the bottom, the UIM automatically calls the program identified by the program dialog variable parameter of the Set List Attributes (QUISETLA) API to acquire more list entries. The program is called repeatedly until either the requested number of entries is added to the list or the list is marked complete at the bottom. For lists that are incomplete at the top, printing begins with the first entry in the list.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Print panel name
INPUT; CHAR(10)
The name of the print head panel or print panel defined in the panel group for the open application.

Eject option
INPUT; CHAR(1)
Determines whether or not this panel begins on a new page. An automatic page eject is done when a print head panel is printed after a print panel. However, even if Y is specified on the QUIPRTP API when the next print panel is printed, it does not cause a second page eject.

One of the following values must be used:

Y The panel is printed at the top of a new page.
N The panel is not always printed at the top of a new page.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A1F E An active display already exists for this application.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A13 E Application &3 closed prematurely.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A18 E Print heading must be specified first.
CPF6A19 E Prologue is only allowed in first heading.
CPF6A23 E Page length too small to print the list column headings.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3B E Application not open for print.
CPF6A3E E Application not open for display.
CPF6A3F E &4 was not found in panel group &1.
CPF6A50 E Error was found during display file or print file operation.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Put Dialog Variable (QUIPUTV) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Input	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	Error code	I/O	Char(*)

The Put Dialog Variable (QUIPUTV) API updates the value of one or more dialog variables by specifying variable buffer of the application program and naming the variable record defined in the panel group for the open application.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
INPUT; CHAR(*)
The program buffer from which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record, which is named in the variable record name parameter.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer. The buffer must be large enough to contain all the dialog variables in the definition of the variable record, which is specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied from the variable buffer into the application variable pool. The variable record must be defined in the panel group for the open application.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A36 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A37 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Remove List Entry (QUIRMVLE) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	List name	Input	Char(10)
3	Extend option	Input	Char(1)
4	List entry handle	Output	Char(4)
5	Error code	I/O	Char(*)

The Remove List Entry (QUIRMVLE) API removes the list entry identified by the value of the current entry pointer for the list. The current entry pointer is always updated to the entry before the one removed. If the first list entry is removed, the current entry pointer is set to the top of the list if the list is complete at the top.

If the list is incomplete at the top, the UIM calls the incomplete list extension program to add another entry to the list.

If the list entry identified by the display position attribute parameter of the Set List Attributes (QUISETLA) API is removed, the display position attribute is set at the entry before the one that was removed. If the new display position attribute is the first entry in the list, the display position attribute is set at the top of the list (logically the entry before the first entry in the list) if the list is complete at the top.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

List name
INPUT; CHAR(10)
The name of the list from which an entry is removed. If the list is not currently active in the open application, an error message is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) API, Add List Multiple Entries (QUIADDLM) API, or its attributes are set with the QUISETLA API.

Extend option
INPUT; CHAR(1)
Specifies whether or not an incomplete list is automatically extended in an attempt to remove the first entry from a list that is incomplete at the top. One of the following values must be specified:

- **Y** The list is extended, if necessary, to add at least one entry to the top of the list or to mark the list as complete at the top.
- **N** The list is not extended to find the entry before the entry being removed, and the entry is not removed from the list.

List entry handle
OUTPUT; CHAR(4)
The list entry handle value from the current entry pointer. A list entry handle uniquely distinguishes an entry until it is removed from the list, even if other entries are inserted and removed from the list. A value of TOP indicates that the current entry pointer is positioned at the top of the list.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A13 E Application &3 closed prematurely.
CPF6A14 E Program defined by variable &4 cannot be called.
CPF6A15 E Errors occurred in list exit program.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A27 E Value for Extend Option parameter not valid.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF6A93 E Operation not valid when current entry is &5.
CPF6A94 E Incomplete list &4 requires extension.
CPF6A95 E List &4 either not complete or not extended.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Remove Pop-Up Window (QUIRMVPW) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Remove option	Input	Char(1)
3	Error code	I/O	Char(*)

The Remove Pop-Up Window (QUIRMVPW) API removes the pop-up window created by the Add Pop-Up Window (QUIADDPW) API. After calling the QUIRMVPW API, subsequent calls to the Display Panel (QUIDSPP) API display either a full-screen panel or a panel in a pop-up window defined by a previous call of the QUIADDPW API. A pop-up window cannot be removed if an action or exit program is currently being processed for a panel displayed in that pop-up window.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Remove option
INPUT; CHAR(1)
The pop-up windows that are removed from the open application. One of the following values must be used:

I All inactive pop-up windows. A pop-up window is inactive if there are no panels currently displayed in that window.

L The most recently added pop-up window.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3E E Application not open for display.
CPF6A84 E Window cannot be removed at this time.
CPF6A90 E Value not correct. Reason code &3.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Remove Print Application (QUIRMVPA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Close option	Input	Char(1)
3	Error code	I/O	Char(*)

The Remove Print Application (QUIRMVPA) API stops print functions in a previously opened display application. The Add Print Application (QUIADDPA) API and the QUIRMVPA API are used in pairs to add and remove printing from display applications.

The QUIRMVPA API closes the printer file for the application. If the QUIRMVPA API is not performed before the application is closed, the Close Application (QUICLOA) API performs the QUIRMVPA API.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Close option
INPUT; CHAR(1)
Specifies whether or not to perform a normal or abnormal close operation on the printer file. One of the following values must be specified:

A An abnormal close operation is performed, and the trailer message is not printed.

M A normal close operation is performed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A1B E Application does not have an open print file.
CPF6A11 E Value is not correct. Reason code is &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve List Attributes (QUIRTVLA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	List name	Input	Char(10)
3	Receiver	Output	Char(*)
4	Receiver length	Input	Binary(4)
5	Error code	I/O	Char(*)

The Retrieve List Attributes (QUIRTVLA) API retrieves the following list attributes:

- The list contents attribute, indicating whether or not all entries are present in the list, and which entries are missing if it is incomplete
- The name of the dialog variable that identifies the program called when the UIM needs to add entries to an incomplete list

- The display position attribute, which is the list entry handle for the entry presented at the top-most row of any list area that displays a list
- The allow trim attribute, which indicates whether or not the UIM trims a full list when adding new entries

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or by the Open Print Application (QUIOPNPA) API when the application is opened.

List name
INPUT; CHAR(10)
The name of the list whose attributes are retrieved. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) or Add List Multiple Entries (QUIADDLM) API, or the fist time the list's attributes are set with the Set List Attributes (QUISETLA) API.

Receiver
OUTPUT; CHAR(*)
The current attributes of the list. For the format of the receiver variable, see "Format of Data Returned."

Receiver length
INPUT; BINARY(4)
The amount of data the application program is prepared to receive. If the length specified is larger than the amount of data available, the receiver is not changed beyond the amount of data available.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of Data Returned

The format of the data available, returned in the receiver parameter, is as follows:

CHAR(4) The list contents attribute, indicating whether or not the list contains all or some of the entries available for display or printing. For more information about the meaning of each possible return value, see the "Set List Attributes (QUISETLA) API" on page 58-27. The following values can be returned in this parameter:

- **ALL** The list is complete. This value is returned when either the QUISETLA API has not been called to set the list attribute or when ALL is the last list contents attribute specified on a call to the QUISETLA API for the list.
- **TOP** Only the top part of an incomplete list is available. This value is returned when TOP is the last list contents attribute specified on a call to the QUISETLA API for the list.
- **BOT** Only the bottom part of an incomplete list is available. This value is returned when BOT is the last list contents attribute specified on a call to the QUISETLA API for the list.
- **MORE** Only the middle part of an incomplete list is available. This value is returned when MORE is the last list contents attribute specified on a call to the QUISETLA API for the list.

CHAR(10) The name of the dialog variable identifying the program the UIM calls when more entries are needed in an incomplete list.

CHAR(4) The display position attribute, which is the list entry handle at the top of the list area on the next panel that displays this list. The UIM does not use the display position attribute for print applications.

The value returned is the handle for a specific entry in the list, or one of the following special values:

- **TOP** The top entries in the list are displayed.
- **BOT** The bottom entries in the list are displayed.

If the list entry identified by the display position attribute of a list is removed before the list is displayed on a panel, the UIM adjusts the attribute to display the entry before the one that was removed. The value returned by this API is not meaningful if entries are removed from the list after the display position attribute is retrieved.

CHAR(1) The allow trim attribute, which indicates whether or not the UIM trims the list when a new list entry causes the list to exceed its maximum size. For additional details, see the "Set List Attributes (QUISETLA) API" on page 58-27.

One of the following values is returned:

- **Y** The UIM automatically trims the list.
- **N** The UIM does not automatically trim the list.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Set List Attributes (QUISETLA) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	List name	Input	Char(10)
3	List contents	Input	Char(4)
4	Program dialog variable	Input	Char(10)
5	Display position attribute	Input	Char(4)
6	Allow trim attribute	Input	Char(1)
7	Error code	I/O	Char(*)

The Set List Attributes (QUISETLA) API sets the following list attributes:

- The list contents attribute, which indicates whether or not all entries are present in the list and which entries are missing if it is incomplete
- The name of the dialog variable that identifies the program the UIM calls when additional entries are needed in an incomplete list
- The display position attribute, which is the list entry handle for the entry presented at the top row of a list area displaying the list
- An attribute indicating whether or not the UIM trims the list if the maximum list size is exceeded when adding a new entry to the list

If the QUISETLA API has not been called for a list since the first entry was inserted, the list contents default value is ALL and the list is complete. An error message is displayed if the user attempts to page beyond either end of the list. The UIM does not call a program for more entries or return to the program that called the Display Panel (QUIDSPP) API.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or by the Open Print Application (QUIOPNPA) API when the application is opened.

List name
INPUT; CHAR(10)
The name of the list whose attributes are changed. If the list is not currently active in the open application, it is activated by this API.

List contents
INPUT; CHAR(4)
Indicates whether or not the list contains all or some of the entries available. One of the following values must be specified:

SAME This list contents attribute of the list is not changed.

ALL The entire list has been built. A message is displayed if the user attempts to page beyond the top or the bottom of the list.

TOP Only the top part of the list is built. A message is displayed if the user attempts to page beyond the beginning of the list. If the user attempts to page beyond the bottom of the list, the program specified by the program dialog variable parameter is called by the UIM, with parameters indicating the need for more entries at the bottom of the list.

BOT Only the bottom part of the list is built. A message is displayed if the user attempts to page beyond the bottom of the list. If the user attempts to page beyond the top of the list, the program specified by the program dialog variable parameter is called by the UIM, with parameters indicating the need for more entries at the top of the list.

MORE Only the middle of the list is built. If the user attempts to page beyond either the top or the bottom the list, the program specified by the program dialog variable parameter is called by the UIM, with parameters indicating the need for more entries at the top or bottom of the list.

If the application program marks the list complete in a given direction, the application program cannot subsequently mark the list incomplete in that same direction.

Program dialog variable
INPUT; CHAR(10)
The name of a dialog variable in an open application. This dialog variable contains information identifying the program called by the UIM when more entries must be added to an incomplete list. The dialog variable must be defined and set properly according to the rules used for the CALL dialog command.

For a description of the CALL dialog command, see the *Guide to Programming Displays*. For a description of the interface between the UIM and the incomplete list exit program, see Chapter 59, "User Interface Management Exit Programs" on page 59-1.

The following special value can be used:

***SAME** The program dialog variable is not changed.

Display position attribute
INPUT; CHAR(4)
The list entry that should be positioned at the top of the list area on the next panel that displays this list. The UIM does not use the display position attribute when printing the list.

The value specified must be the list entry handle currently existing in the list or one of the following special values:

SAME The display position attribute of the list is not changed.

TOP The entries at the top of the list should be positioned at the top of the list area on the next panel.

BOT The bottom *n* entries of the list should be displayed in the list area on the next panel, where *n* is the number of list entries that can be displayed in the current view of the list. If the list contains fewer than *n* list entries, the complete list is displayed.

If the list entry identified by the display position attribute of a list is removed before the list is displayed on a panel, the UIM automatically adjusts the attribute to display the entry before the one removed. If there is no previous entry in the list, the attribute is set at the top of the list, and the next panel displaying the list shows the first entry in the list.

During incomplete list extension, the application program must specify SAME for the display position attribute.

Allow trim attribute
INPUT; CHAR(1)
Indicates whether or not the UIM should trim the list during the Add List Entry (QUIADDLE) or the Add List Multiple Entries (QUIADDLM) API if adding the new entry exceeds the maximum list size. The UIM trims only lists that have been marked as incomplete by using the QUISETLA API. When the UIM removes the list entry, the UIM marks the list incomplete in the direction from which the entry was removed, regardless of whether or not the list was previously incomplete in this direction.

If an application is adding an entry to the bottom of the list when the maximum size list is reached, the UIM removes the first list entry and marks the list incomplete at the top. The incomplete list exit program of the application must be prepared to add list entries in either direction of an incomplete list.

When the incomplete list exit program is called to add list entries, the QUIADDLE or QUIADDLM API can be called to add entries in the middle of the list. If the entries are added to the middle of the list when the list is complete; trimming does not take place and an error is reported indicating that the maximum list size has been reached.

One of the following values must be specified:

S The allow trim attribute of the list should not be changed.

Y The UIM automatically trims the list.

N The UIM does not automatically trim the list.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA0 E Request is not allowed when extending a list that is not complete.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A9A E Dialog variable required to specify incomplete list.
CPF6A9B E Dialog variable &5 not a pointer variable.
CPF6A9C E List entry identifier not correct.
CPF6A9E E Value for Allow Trim parameter not valid.
CPF6A9F E Attribute for list &4 not valid.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A99 E Dialog variable &5 not found in panel group &1 in &2.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Set Screen Image (QUISETSC) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Error code	I/O	Char(*)

The Set Screen Image (QUISETSC) API establishes the screen image for a UIM application. This API is used when a pop-up window is displayed over a panel that was not displayed using the same UIM application.

For displaying pop-up windows over data description specifications (DDS) screens, it is the responsibility of the application program to save and restore the display before and after displaying the pop-up window. This can be done by specifying RSTDSP(*YES) for the DDS display file.

When the QUISETSC API is called, a read screen command is sent to the requesting program device. The screen image

returned from the read screen operation is saved for the open application. Subsequent panels displayed in pop-up windows are displayed on top of this image. Using this API to set the screen image is functionally similar to using the Display Panel (QUIDSPP) API to establish the primary panel on which pop-up windows are overlaid.

The UIM does not preserve the extended character buffer (ECB) attributes on the underlying panel. This means that the UIM issues a read screen command instead of a read screen with ECB command. Any highlighting on the underlying panel that was specified using ECB attributes is lost. When the application redisplays the underlying panel after the pop-up window is removed, the correct highlighting is restored.

No half-index ECB attributes are preserved.

Any double-byte character string (DBCS) data in the underlying panel is supported, including DBCS data whose shift-in and shift-out characters are specified in ECB attributes.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API when the application is opened.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A1F E An active display already exists for this application.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A3E E Application not open for display.
CPF6A50 E Error was found during display file or print file operation.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Update List Entry (QUIUPDLE) API

Parameters

Required Parameter Group:

1	Application handle	Input	Char(8)
2	Variable buffer	Input	Char(*)
3	Variable buffer length	Input	Binary(4)
4	Variable record name	Input	Char(10)
5	List name	Input	Char(10)
6	Option	Input	Char(4)
7	List entry handle	Output	Char(4)
8	Error code	I/O	Char(*)

The Update List Entry (QUIUPDLE) API updates the list entry identified by the current entry pointer for the list or the extended action entry. The current contents of all dialog variables corresponding to dialog variables in the list are saved in the entry. The current entry pointer of the list is not changed by this operation.

Required Parameter Group

Application handle
INPUT; CHAR(8)
The application handle assigned by the UIM and returned to the application program by the Open Display Application (QUIOPNDA) API or the Open Print Application (QUIOPNPA) API when the application is opened.

Variable buffer
INPUT; CHAR(*)
The program buffer from which dialog variable values are copied. The dialog variables are copied in the order specified in the variable record definition.

If the variable record name parameter specifies the name of a variable record, which is defined in the panel group for this open application, dialog variables are copied from the variable buffer to the application variable pool before the list entry is updated. The operation of this parameter is the same as using the Put Dialog Variable (QUIPUTV) API immediately before the QUIUPDLE API.

The variable buffer must be large enough to contain all the variables specified in the variable record definition.

Variable buffer length
INPUT; BINARY(4)
The length of the variable buffer provided. The buffer must be large enough to contain all the dialog variables in the definition of the variable record, specified in the variable record name parameter.

Variable record name
INPUT; CHAR(10)
The name of the variable record that determines which dialog variables are copied between the variable buffer and the application variable pool. The variable record must be defined in the panel group for the open application.

The following special value can be used:

***NONE** The QUIPUTV API is not done during the QUIUPDLE API. The variable buffer parameter is ignored when this value is used.

List name
INPUT; CHAR(10)
The name of the list in which an entry is updated. If the list is not currently active in the open application, an error is reported. A list is made active the first time an entry is inserted with the Add List Entry (QUIADDLE) API or the Add List Multiple Entries (QUIADLM) API, or its attributes are set with the Set List Attributes (QUISETLA) API.

Option
INPUT; CHAR(4)
The updated list entry. One of the following values must be specified:

EXTE The extended action entry is updated.

SAME The list entry identified by the current entry pointer is updated.

List entry handle
OUTPUT; CHAR(4)
The list entry handle for the updated list entry. A list entry handle uniquely distinguishes an entry until it is removed from the list, even if other entries are inserted and removed from the list.

When the option parameter has the value EXTE, this parameter returns the value EXTE, indicating that the extended action entry is updated.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF6AA1 E The value of the action field is not correct at this time. Reason code &5.
CPF6A0B E Application identifier &3 not valid.
CPF6A0C E Application domain error for application &1.
CPF6A0F E Previous error occurred while running application &3.
CPF6A24 E Parameter &1 not passed correctly.
CPF6A25 E Return code length of &1 not valid.
CPF6A28 E Value for Option parameter not valid.
CPF6A36 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A37 E Data not correct for dialog variable &4 in panel group &1 in &2.
CPF6A38 E Record &4 not defined in panel group.
CPF6A39 E Buffer length too small.
CPF6A90 E Value not correct. Reason code &3.
CPF6A91 E List &4 does not exist.
CPF6A92 E List &4 not active.
CPF6A93 E Operation not valid when current entry is &5.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Chapter 59. User Interface Management Exit Programs

Through panel definitions and APIs, an application program can cause the UIM to call other application programs as part of normal panel management. These programs are divided into two classes: programs called through the CALL dialog command and programs called as exit programs.

This chapter describes the different exits and CALL situations. The UIM does not require a program to follow prescribed actions in its operating environment; attempts to use these exit programs for purposes other than the intended purpose may cause problems for applications and their users.

The CALL dialog command is provided so that an application can link programs and panels without a command interface.

The CALL dialog command causes program calls to handle one of the following requests:

- Function keys
- Menu items
- Action list options
- Pull-down field choices

Exit programs allow applications to perform functions for which the UIM does not provide generic support. Exit program capability exists for the following purposes:

- General panel checking
- Action list update processing
- Incomplete list processing
- Application formatting of data
- Cursor-sensitive prompting

Calling an Exit Program

All programs are specified by naming a dialog variable, which identifies the program to call. For the incomplete list exit program, the dialog variable name is specified on the call to the Set List Attributes (QUISETLA) API. For all other exit and CALL programs, the dialog variable name is specified as a parameter to the CALL dialog command in the tag language source.

The UIM uses the current value of the dialog variable to identify the program to call. Depending on the definition of the dialog variable, there are three different ways to call the program:

Call by address
When the dialog variable specified is defined with BASETYPE=PTR on the class definition (CLASS) tag, the program is called by address.

It is the responsibility of the application program to ensure that the pointer is set correctly.

Call by name
When the dialog variable specified is defined with BASETYPE='CHAR 20' on the CLASS tag, the program is called by name. It is the responsibility of the application program to ensure that the variable contains the object and library name of the program called.

Extended program model (EPM) call
When the dialog variable specified is defined with BASETYPE='CHAR 130' on the CLASS tag, the program is called in the extended program model (EPM) environment. This type of call is used when the target program is written in an EPM language such as Pascal or C/400. It is the responsibility of the application program to ensure that the variable contains the appropriate information for calling an entry point for an EPM environment.

The UIM can call programs in the Integrated Language Environment (ILE) model using call by address or call by name. Only the main entry point of the program can be used.

Using a Parameter Interface

When a UIM application is opened, the calling program must specify whether all application programs, called as an exit program or using the CALL dialog command, are passed a single or multiple parameter interface.

When a single parameter is passed, that parameter is a space pointer to a structure containing information for the type of exit or CALL function.

The single parameter interface is used by programs written in languages that can efficiently process data structures. Passing a single parameter is more efficient than passing multiple parameters, and any extensions to the interface are accessible without changing the parameter list of the program being called.

When the single parameter interface is chosen, the structure passed contains a field indicating the level of the structure. Beginning in Version 2 Release 2 (V2R2) of the OS/400 operating system, the structure level is set to two. All parameters and values of a structure are passed when the structure level is equal to or greater than one, unless the parameter description notes otherwise. For more information about choosing the interface level for exit programs, see "Open Display Application (QUIOPNDA) API" on page 58-18 and "Open Print Application (QUIOPNPA) API" on page 58-20.

The interface with multiple parameters is available for programs written in languages that cannot efficiently process data structures. When multiple parameters are passed, each parameter is a space pointer to a piece of information necessary for the type of exit or CALL function.

When the multiple parameter interface is chosen by the program opening the UIM application, the program must also choose which interface level to use. The **interface level** defines the number of parameters passed for a given type of

exit or CALL function. This interface allows the parameter interfaces to be extended in the future with upward compatibility for existing application exit programs.

Because the application handle is passed, programs can use dialog variables to access common information. It is more efficient to place one pointer variable that points to a structure or array into the variable pool than to place all the variables from the structure or array into the variable pool.

Handling Messages

After the UIM calls an application program or runs a control language (CL) command, it handles messages sent to the UIM. Messages sent by the general exit program are moved to an internal UIM program message queue. These messages are referred to as **transient** messages. When the panel is redisplayed by the UIM, the transient messages are shown on the message line of the panel. When the next operation is performed for the panel, in most cases, transient messages are removed from the UIM's internal program message queue and no longer appear in the job log.

Messages sent by any other type of exit program or by a CL command are moved to the program message queue identified on the call to the Display Panel (QUIDSPP) API. These messages are referred to as **user** messages. When the panel is redisplayed by the UIM, the user messages are shown on the message line of the panel. When certain dialog commands are subsequently performed for the panel, the UIM no longer displays the user messages on the panel. It is the responsibility of the application program to remove these messages from the program message queue as appropriate. For more information about when the UIM stops displaying the user messages for a panel, see the *Guide to Programming Displays*.

Escape messages and other accompanying messages are displayed on the panel message line. If the program is sent an escape message to be resent, it needs to do the following:

1. Receive the exception message and save its message reference key.
2. Move all INFO, COMP, and DIAG messages to the previous program message queue.
3. Perform any cleanup processing required for the application program.
4. Resend the exception message using its message reference key.

Whenever an escape message is sent to the UIM from the program or a CL command, the message indicates that the function performed was unsuccessful. The UIM then takes whatever action is appropriate for the unsuccessful function.

CALL Program for a Function Key

Function keys can be assigned to the CALL dialog command on the ACTION attribute of the key item (KEYI) tag. When the function key is pressed, the UIM calls the specified program.

The CALL dialog command is assigned to a function key to process requests that are specific to the application. These requests either do not have a command interface or require more knowledge about the current application than can normally be passed through a command.

The application can perform most functions, but the application developer must be aware of the effects of changing dialog variables, displaying other panels, and so on.

Single Parameter Interface

Structure level
BINARY(4); positions 1–4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5–12

Type of call
BINARY(4); positions 13–16
Set to the following value:

1 Processes a function key.

Application handle
CHAR(8); positions 17–24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25–34
The name of the panel currently being processed by the UIM.

Function key pressed
BINARY(4); positions 35–38
The function key that is pressed:

1–24 A function key (F1–F24) is processed.
26 The default action when the Enter key is pressed.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Function key pressed	Input	Binary(4)

No additional parameters are required for interface level 2.

CALL Program for a Menu Item

Menu items can be assigned to the CALL dialog command on the ACTION attribute of the menu item (MENUI) tag. When the menu item is selected, the UIM calls the specified program.

The CALL dialog command is assigned to a menu item to process requests that are specific to the application. These requests either do not have a command interface or require more knowledge about the current application than can normally be passed through a command.

The application can perform most functions, but the application developer must be aware of the effects of changing dialog variables, displaying other panels, and so on.

Single Parameter Interface

Structure level
BINARY(4); positions 1–4
The interface level supported by this structure, indicating which fields and values are available.

Reserved
CHAR(8); positions 5–12

Type of call
BINARY(4); positions 13–16
Set to the following value:

2 Processes a menu item.

Application handle
CHAR(8); positions 17–24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25–34
The name of the panel currently being processed by the UIM.

Menu option
BINARY(4); positions 35–38
The option number of the menu item selected by the user.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Menu option	Input	Binary(4)

No additional parameters are required for interface level 2.

CALL Program for an Action List Option and Pull-Down Field Choice

An action list option can specify the CALL dialog command on the ENTER, EXTENTER, PROMPT, and EXTPROMPT attributes of the list action (LISTACT) tag. When processing an action list, the UIM calls the specified program.

A pull-down choice can specify the CALL dialog command on the ACTION attribute of the pull-down field choice (PDFLDC) tag. When processing a selected pull-down choice, the UIM calls the specified program.

The CALL dialog command on an action list or pull-down choice with ACTFOR=LIST on the PDFLDC tag allows the application to handle a request for a single list entry. When processing the list, the UIM calls the program once for each selected entry in the list. Care must be taken in changing other list entries and the current position because the UIM is accessing the same information to perform the remaining action list processing. For example, changing the current position past some selected list entries causes the UIM to not process the skipped ones.

The CALL dialog command on a pull-down choice with ACTFOR=PANEL on the PDFLDC tag allows the application to handle a request related to the panel as a whole. The UIM calls the program only once before redisplaying the panel.

An exception message received by the UIM while performing list processing stops the processing and displays the messages. The action or selection field still contains the option number or selection character and is marked in error.

Single Parameter Interface

Structure level
BINARY(4); positions 1–4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5–12

Type of call
BINARY(4); positions 13–16
Set to one of the following values:

3 A program is called for action list option processing or pull-down choice processing when ACTFOR=LIST is specified on the PDFLDC tag.

9 A pull-down choice. A program is called for pull-down choice processing when ACTFOR=PANEL is specified on the PDFLDC tag.

Application handle
CHAR(8); positions 17–24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25–34
The name of the panel currently being processed by the UIM.

List name
CHAR(10); positions 35–44
The name of the list currently being processed by the UIM. Blanks indicate that the pull-down choice being processed does not operate against a list entry because ACTFOR=PANEL is specified on the PDFLDC tag.

List entry handle
CHAR(4); positions 45–48
The list entry handle currently being processed by the UIM. Hexadecimal zeros indicate that the pull-down choice being processed does not operate against a list entry.

The following value may be used:

EXTE Processes the extended action entry.

Option number
BINARY(4); positions 49–52
The option number of the list action or pull-down choice being processed.

Function qualifier
BINARY(4); positions 53–56
One of the following values:

0 Performs the action specified on the ENTER or EXTENTER attribute of the LISTACT tag or the ACTION attribute of the PDFLDC tag.

–10 Performs the action specified on the PROMPT or EXTPROMPT attribute of the LISTACT tag.

Pull-down field name
CHAR(10); positions 57–66
The name of the pull-down field from which the pull-down choice is selected. If this pull-down field does not have a name, this field is set to blanks.

This field is only available when the field for the structure level is set to two or greater.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	List name	Input	Char(10)
5	List entry handle	Input	Char(4)
6	Option number	Input	Binary(4)
7	Function qualifier	Input	Binary(4)

Additional parameter for interface level 2:

8	Pull-down field name	Input	Char(10)

Exit Program for General Panel Checking

A general exit program may be specified on the USREXIT attribute of the panel definition (PANEL) tag. This attribute specifies the name of a dialog variable identifying the program to call.

The UIM calls the program:

- After any necessary validity checking is done by the UIM

  For a VARUPD=YES attribute on the key item (KEYI) or pull-down field choice (PDFLDC) tags, the UIM has already performed validity checking on all displayed values. If errors are found, the UIM does not call the exit program. The current panel is automatically redisplayed with the appropriate message and error highlighting.

  For a VARUPD=NO attribute, the UIM saves all values from entry fields in their displayed form and does not update the dialog variables. The exit program does not have access to these values.

- After the UIM determines what function to perform

  For some dialog commands, the UIM does not call the exit program. For more information on when the UIM

calls the exit program, see *Guide to Programming Displays*.

Any errors detected during function determination cause the UIM to redisplay the panel without calling the exit program.

- Before the UIM performs the determined function, including dialog variable substitution in command strings specified in the CMD dialog command.

The UIM calls the general exit program and passes information about the function. The exit program performs its task and returns control to the UIM through a normal return or by sending a CPF6A02 or CPF6A03 status message.

If the exit program returns control by sending a CPF6A02 status or escape message, the UIM does not perform the determined function. The panel is redisplayed and any previous messages sent to the UIM are shown.

If the exit program returns control by sending a CPF6A03 status or escape message, the UIM continues performing the determined function. Any previous messages sent to the UIM by the general exit program are shown if and when the function completes.

If the exit program returns control normally, the UIM continues performing the determined function. Any messages sent to the UIM by the exit program are ignored.

Messages sent to the UIM are displayed only once. They are cleared when the next dialog command is performed, excluding the HELP, PAGEUP, PAGEDOWN, and the PRINT dialog commands. If the exit program marks dialog variables in error, the exit program or other application code is responsible for resetting the dialog variables when the variable is no longer incorrect.

The general exit program allows applications to perform extended validity checking on field values. However, other functions are possible because the exit program has access to panel information used by the UIM to perform a specified function. The exit program can prevent the UIM from processing the function. For example, the exit program could intercept a function key, perform some other function, and tell the UIM to not process the function key. While the UIM does not prevent this type of thing, it is the application developer's responsibility to understand enough about UIM processing to ensure that things work properly.

The action performed by the UIM is determined before the exit program is called. Some actions, such as whether or not to perform the action list processing, depend on the contents of dialog variables or list entries. Changes to these variables or list entries do not cause the UIM to evaluate the determined action again.

Dialog variable substitution into command strings is not done until after the exit program is called; changing dialog variables can affect the final, submitted command.

The exit and cancel flags for the job are not reset before the exit program is called, and they are not checked after the exit program returns control to the UIM. Therefore, exit and cancel flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1–4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5–12

Type of call
BINARY(4); positions 13–16
Set to the following value:

4 Processes a general panel exit.

Application handle
CHAR(8); positions 17–24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25–34
The name of the panel currently being processed by the UIM.

Function key pressed
BINARY(4); positions 35–38
The function key that is pressed:

1–24 A function key (F1–F24) is pressed.
26 The Enter key is pressed.
28 The Page Down key is pressed.
29 The Page Up key is pressed.
31 The Home key is pressed.

The general exit program is not called for any other keys.

Function key qualifier
BINARY(4); positions 9–42
Provides information about the function that will be performed unless the exit program returns through a CPF6A02 message:

–1 Submits a command from the command line.

–2 Performs list action processing. The ACTOR attribute of the list area (LIST) tag determines whether the processing is to be performed by the UIM or by the application program displaying the panel.

–3 Processes a menu item selection.

–4 The Enter key is pressed and there is no function for the UIM to perform. The default enter action specified on the ENTER attribute of the display panel (PANEL) tag will be performed.

–5 Processes a cursor-sensitive prompt function.

–6 Processes a selection from a pull-down choice. This value is used only when the structure level is two or greater.

–7 Processes the default selection action specified on the SELECT attribute of the PANEL tag because the Enter key was pressed or the user selected one or more entries in an action list or selection list, and there was no function for the UIM to perform.

Option number
BINARY(4); positions 43–46
When the field for the function key qualifier indicates that the UIM is processing a menu item, this field is set to the option number of the menu item selected by the user.

When the field for the function key qualifier indicates that the UIM is processing a pull-down choice, this field is set to the option number of the pull-down choice selected by the user.

For all other values for the function key qualifier, this field is not set.

Pull-down field name
CHAR(10); positions 47–57
The name of the pull-down field from which the pull-down choice is selected. If this pull-down field does not have a name, it is set to blanks.

For all other values of the function key qualifier, this field is not set.

This field is only available when the field for the structure level is set to two or greater.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Function key pressed	Input	Binary(4)
5	Function key qualifier	Input	Binary(4)
6	Option number	Input	Binary(4)

Additional parameter for interface level 2:

7	Pull-down field name	Input	Char(10)

Exit Program for an Action List Option or Pull-Down Field Choice

An exit program can be specified for an action list option using the USREXIT attribute of the list action (LISTACT) tag. The exit program is called immediately after the action specified on the ENTER, PROMPT, EXTENTER, or EXTPROMPT attribute of the LISTACT tag is processed.

An exit program for a pull-down field choice can be specified using the USREXIT attribute of the pull-down field choice (PDFLDC) tag. The exit program is called immediately after the action specified on the ACTION attribute of the PDFLDC tag is processed.

Normally, this exit program adds, updates, or removes a list entry for the application when the action list option or pull-down choice action is a command string. The exit program should not change list entries other than the one currently being processed. If it does, remaining UIM list processing may not perform as expected. If the exit program changes the current entry for the list by adding a new entry to the list, the exit program should reset the current entry to the one currently being processed before returning control to the UIM. If the exit program does not do this, remaining UIM list processing may not perform as expected.

Exception messages received by the UIM from the exit program cause the UIM to stop list processing, and display the messages when the panel is reshown. If the dialog variable identifying the exit program is null (for a pointer variable) or blanks (for a character variable), no exception is reported by the UIM and processing continues as if the USREXIT attribute is not specified.

The cancel and exit flags for the job are not reset before the exit program is called, and the flags are not checked after the exit program returns control to the UIM. Therefore, the cancel and exit flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1–4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5–12

Type of call
BINARY(4); positions 13–16
Set to the following value:

5 The exit program is called for an action list option or pull-down choice when ACTFOR=LIST is specified on the PDFLDC tag.

Application handle
CHAR(8); positions 17–24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25–34
The name of the panel currently being processed by the UIM.

List name
CHAR(10); positions 35–44
The name of the list currently being processed by the UIM.

List entry handle
CHAR(4); positions 45–48
The handle of the list entry being processed by the UIM. The following value may be used:

EXTE Processes the extended action entry.

Option number
BINARY(4); positions 49–52
The option number of the list action or pull-down choice being processed for the specified list entry.

Function qualifier
BINARY(4); positions 53–56
Set to one of the following values:

0 Performs the action specified on the ENTER or EXTENTER attribute of the list action (LISTACT) tag or the ACTION attribute of the PDFLDC tag.

–10 Performs the action specified on the PROMPT or EXTPROMPT attribute of the LISTACT tag.

Action results
BINARY(4); positions 57–60
Set to one of the following values, indicating whether or not the list action or pull-down choice was successful:

0 The list action or pull-down choice is successful.
1 The list action or pull-down choice is unsuccessful.

A list action or pull-down choice is considered unsuccessful if any of the following occurs:

- An exception message is sent by the action program or command.
- The cancel flag for the job is set on by the action program or command.
- The exit flag for the job is set on by the action program or command.
- The list action or the pull-down choice sent the CPF6A01 status or escape message to stop list action processing.

Conversely, the list action is successful if none of the above occur.

Pull-down field name
CHAR(10); positions 61–70
The pull-down field name from which the pull-down choice is selected. If the pull-down choice does not have a name, this field is set to blanks.

When the action being processed is not a pull-down choice, this field is set to blanks.

This field is available only when the structure level field is set to two or greater.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	List name	Input	Char(10)
5	List entry handle	Input	Char(4)
6	Option number	Input	Binary(4)
7	Function qualifier	Input	Binary(4)
8	Action results	Input	Binary(4)

Additional parameter for interface level 2:

9	Pull-down field name	Input	Char(10)

Exit Program for an Incomplete List

An exit program for handling an incomplete list is defined using the Set List Attributes (QUISETLA) API. A dialog variable is specified identifying the program to call.

The dialog variable cannot be blanks or nulls. The UIM cannot process an incomplete list without the exit program.

This exit allows an application to display part of a list without having to build the entire list. It must either add more entries or mark the list as complete in the direction being extended. If the list is not marked complete and fewer entries than the minimum requested by the UIM are added, the exit program is called again. If the list is not marked complete and no entries are added by the exit program, a CPF6A95 error is reported to the program that called the Display Panel (QUIDSPP), Get List Entry (QUIGETLE), Get List Multiple Entries (QUIGETLM), or Remove List Entry (QUIRMVLE) API.

The exit program can send messages to the UIM for display on the panel. To have the messages shown, the exit program must return control to the UIM by sending a

CPF6A05 status or escape message. In this case, any messages sent to the UIM by the incomplete list exit program are shown when the panel is displayed. Otherwise, the exit program returns control to the UIM by doing a normal return.

Sometimes the exit program for an incomplete list is called when no panel is displayed; such a call occurs when the QUIGETLE, the QUIGETLM, or the QUIRMVLE API is used. If no panel is displayed when the exit program sends a CPF6A05 status or escape message, any messages sent to the UIM by the exit program are moved to the program message queue for the program calling that API.

The exit and cancel flags for the job are not reset before the exit program is called, and the flags are not checked after the exit program returns control to the UIM. Therefore, the exit and cancel flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1–4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5–12

Type of call
BINARY(4); positions 13–16
Set to the following value:

6 Processes an incomplete list exit program.

Application handle
CHAR(8); positions 17–24
The application handle of the application currently being processed by the UIM.

Reserved
CHAR(10); positions 25–34

List name
CHAR(10); positions 35–44
The name of the list currently being processed by the UIM.

Incomplete list direction
BINARY(4); positions 45–48
The direction from the current list entry in which additional entries must be added.

Set to one of the following values:

0 More list entries are required after the current list entry.
1 More list entries are required before the current list entry.

Number of entries required
BINARY(4); positions 49–52
The minimum number of entries that must be added to the list.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	List name	Input	Char(10)
4	Incomplete list direction	Input	Binary(4)
5	Number of entries required	Input	Binary(4)

No additional parameters are required for interface level 2.

Exit Program for Application Formatted Data

The exit program can update the data formatted by the application every time a panel is displayed, and returns control to the UIM through a normal return. The UIM then finishes formatting the panel and displays it. The area formatted by the application is displayed as defined by the contents of the dialog variable.

An application format exit program can be specified on the USREXIT attribute of the application formatted area (APPFMT) tag. This attribute specifies the name of a dialog variable identifying the program to call.

This exit program is called during the formatting of the panel whenever the panel is displayed. If the dialog variable identifying the program is nulls for a pointer variable or blanks for a character variable, no exception is reported and processing continues as if the USREXIT attribute is not specified.

The exit and cancel flags for the job are not reset before the exit program is called, and the flags are not checked after the exit program returns control to the UIM. Therefore, the exit and cancel flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1–4
The interface level supported by this structure, indicating

which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5–12

Type of call
BINARY(4); positions 13–16
Set to the following value:

7 The exit program is called to update the application formatted data.

Application handle
CHAR(8); positions 17–24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25–34
The name of the panel currently being processed by the UIM.

Panel bidirectional orientation
CHAR(1); positions 35–35
Identifies the attribute orientation of the panel group.

Set to one of the following values:

N BIDI=NONE is specified or the default on the panel group (PNLGRP) tag.
L BIDI=LTR is specified on the PNLGRP tag.
R BIDI=RTL is specified on the PNLGRP tag.

Device code page
BINARY(4); positions 36–39
The number of the code page of the requester's display device.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Panel bidirectional orientation	Input	Char(1)
5	Device code page	Input	Binary(4)

No additional parameters are required for interface level 2.

Exit Program for a Cursor-Sensitive Prompt

An exit program for handling a cursor prompt request can be specified on the PROMPT attribute of the data item (DATAI), data item extender (DATAIX), and list column (LISTCOL) tags. This attribute specifies the name of a dialog variable identifying the program to call.

This dialog variable cannot be blanks or nulls. The UIM cannot process a prompt request without the exit program.

Normally, the exit program displays a selection list panel. Such a list panel allows the user to select one or more entries from a list of possible choices. The application then sets the selected choices in the appropriate dialog variables, which are shown when the UIM redisplays the panel where the prompt function was requested.

The exit and cancel flags for the job are not reset before the exit program is called, and the flags are not checked after the exit program returns control to the UIM. Therefore, the exit and cancel flags should not be turned on by the exit program.

Single Parameter Interface

Structure level
BINARY(4); positions 1–4
The interface level supported by this structure, indicating which fields and values are available for the current interface level.

Reserved
CHAR(8); positions 5–12

Type of call
BINARY(4); positions 13–16
Set to the following value:

8 The exit program is called to provide cursor-sensitive prompting.

Application handle
CHAR(8); positions 17–24
The application handle of the application currently being processed by the UIM.

Panel name
CHAR(10); positions 25–34
The name of the panel currently being processed by the UIM.

Panel element type
CHAR(1); positions 35–35
One of the following values identify the type of panel element prompt:

1 Prompts for a data item (DATAI tag).
2 Prompts for a data item extender (DATAIX tag).
3 Prompts for a list column (LISTCOL tag).

Reserved
CHAR(1); positions 36–36

Dialog variable name
CHAR(10); positions 37–46
The name of the dialog variable where the cursor was positioned when the prompt function was requested.

List name
CHAR(10); positions 47–56
The name of the list where the cursor was positioned when the prompt function was requested. This field is set only when the panel element type field indicates a LISTCOL tag is prompted.

List entry handle
CHAR(4); positions 57–60
The list entry handle where the cursor was positioned when the prompt function was requested. This field is set only when the panel element type field indicates a LISTCOL tag is prompted.

Multiple Parameter Interface

The description of parameters listed in the following table is the same as the corresponding field in the structure for a single parameter interface. The positions listed for each field in the structure do not apply to the multiple parameter interface.

Parameters

Parameters for interface level 1:

1	Type of call	Input	Binary(4)
2	Application handle	Input	Char(8)
3	Panel name	Input	Char(10)
4	Panel element type	Input	Char(1)
5	Dialog variable name	Input	Char(10)
6	List name	Input	Char(10)

Additional parameter for interface level 2:

7	List entry handle	Input	Char(4)

Part 23. Virtual Terminal APIs

Chapter 60. Introduction to Virtual Terminal APIs

The virtual terminal APIs allow your AS/400 application programs to interact with AS/400 application programs that are performing work station input and output (I/O).

A **virtual terminal** is a device that does not have hardware associated with it. It forms a connection between your application and AS/400 applications, representing a physical work station (possibly on a remote system). The OS/400 licensed program manages the virtual terminal, which directs work station I/O performed by an AS/400 application to the virtual terminal. The virtual terminal APIs allow another AS/400 application, called a **server program**, to work with the data associated with the virtual terminal.

In a distributed systems environment, the requesting program is called a **client**; the answering program is called a **server**. The client and server programs may reside on the same AS/400 system or may be distributed between two different systems. The server program generally runs on behalf of (or in conjunction with) the client program. Together, the server program and the client program allow a work station to be supported as if the work station were connected locally.

Chapter 61, "Virtual Terminal APIs" provides the virtual terminal APIs, including syntax, parameters, notes about usage, and associated AS/400 system messages. Chapter 62, "Virtual Terminal Run-Time Example" contains an example of how the OS/400 operating system, the server program, the client program, and the work station device interact when processing a system request.

Distributed 5250 Emulation Model

Figure 60-1 shows a model for a distributed systems environment between an IBM PS/2* work station and an AS/400 system. The client program resides on the PS/2 work station, while the server program resides on the AS/400 system. This model is similar to the way the AS/400 supports a PC Support/400 work station function.

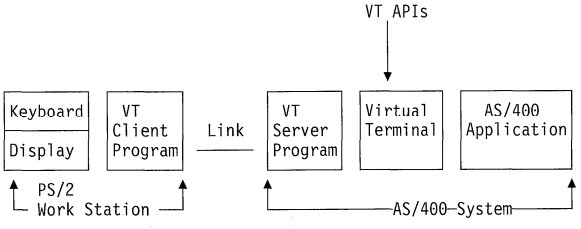


Figure 60-1. Example Virtual Terminal Client/Server Model

The client program running on the PS/2 work station shown in Figure 60-1 does the following:

- Accepts data from the server program and displays the data on the PS/2 display
- Accepts data from the PS/2 keyboard
- Converts the data from the format required by the PS/2 display and keyboard to the format required by the server program (5250 data stream)
- Sends the data to the server program on the AS/400 system

The link between the client program and server program uses DOS and OS/400 communications support. This may be LU 6.2, TCP/IP, or some other communications protocol.

You can write a server application program in any AS/400-supported high-level language (HLL), such as the C/400 programming language. The server program provides work station support for the AS/400 system acting as the server. The virtual terminal APIs allow a server program to read and write virtual terminal data. Virtual terminal data is always in a 5250 data stream format. See the *5250 Functions Reference Manual*, SA21-9247, for more information on 5250 data streams.

The virtual terminal APIs provide an interface between the server program and the virtual terminal supported by the OS/400 licensed program. The virtual terminal represents the OS/400 link between the server program and the AS/400 application and is managed entirely by the OS/400 licensed program.

The AS/400 application is a licensed program or a user-written application that performs a standard data transfer to a work station. This application can be written in any AS/400-supported HLL.

AS/400 Job Information

Several AS/400 jobs are involved when you use the virtual terminal APIs. The jobs can be classified into two groups:

- Server program jobs
- Application jobs

Each group may consist of one or more jobs, depending on the way the server program is written and the way the AS/400 applications are being run.

The server program can be written to run in a single job or in more than one job, depending on the number of work stations to be supported per job. For example, you can support each work station using a single job by routing all requests from the work station client program to a particular server program job.

AS/400 Subsystem Information

A server program should run in the same subsystem that other server programs are running. For controlling resource use, such as main storage, a separate subsystem for running all server programs is usually best. This is also advantageous for allowing performance tuning, such as the number of page faults. Generally, AS/400 applications run in subsystem QBASE.

If the subsystem under which you want the server program to run was created with the system defaults, you will not have to add a work station entry to the subsystem description. However, if you do need to add a work station entry to the subsystem description, you can use the Add Work Station Entry (ADDWSE) command.

Before a work station is allowed to sign-on, it must be defined to a subsystem. In this case, the work station is the virtual terminal device (QPADEVnnnn) automatically created by the OS/400 licensed program. The work station name, work station type, or *ALL must be specified in the subsystem description. Use the Display Subsystem Description (DSPSBSD) command to see the list of work station entries defined for a subsystem. The following command can be used to add all work station types to a subsystem named QBASE:

```
ADDWSE SBSD(QBASE) WRKSTNTYPE(*ALL)
```

For more information on automatically creating virtual terminals, see "Step 1: Setting the Number of Automatically Created Virtual Terminals" on page 60-3.

Note: The ADDWSE command is only valid when the subsystem description is not active.

Work Station Types

You must specify a work station type when a virtual terminal device is opened. A server program can select several different types of work stations. See "Supported Work Station Types and Models" on page 61-2 for a list of the types of work stations that are supported.

Data Queues

The OS/400 licensed program uses data queues to send data to the server program. The server program also uses data queues for interprocess communications with other AS/400 applications and API calls.

The data queue must exist before your application uses the virtual terminal APIs to open a path to a virtual terminal. See the *CL Reference* for details about creating and deleting data queues. See "Open Virtual Terminal Path (QTVOPNVT) API" on page 61-1 for information on how a server program specifies the name of the data queue to be used.

The OS/400 licensed program communicates special events to the server program using the data queue. The AS/400 system provides the information about the special events using a data queue entry.

The following events result in data queue entries being sent:

- The virtual terminal receives data from an AS/400 application
- OS/400 closes the virtual terminal (This occurs when an AS/400 application's job is canceled.)

Figure 60-2 shows the structure of OS/400 data queue entries that are sent to the data queue when an application uses the virtual terminal APIs. All data queue entries have the same format.

Figure 60-2. Format for OS/400 Data Queue Entries

Name	Type	Description
Entry Type	CHAR(10)	Always set by OS/400 to *VRTTRM.
Entry ID	CHAR(2)	Entry ID associated with entry. Valid values for the 2 characters in this parameter are: 1. OS/400 is closing (terminating) the session with the virtual terminal. The server program should perform a close to indicate that the server program is done using the virtual terminal. 2. An AS/400 application has sent data to the virtual terminal. see "Read from Virtual Terminal (QTVRDVT) API" on page 61-3 for information on how to read the data. **Note:** All other values are reserved by AS/400.
VTHandle	CHAR(16)	The virtual terminal handle associated with the virtual terminal communications path previously opened by calling the Open Virtual Terminal Path (QTVOPNVT) API. See "Open Virtual Terminal Path (QTVOPNVT) API" on page 61-1 for additional details.
	CHAR(52)	Reserved
Key	CHAR(*)	Key value specified when the virtual terminal communications path was opened. See "Open Virtual Terminal Path (QTVOPNVT) API" on page 61-1 for additional details on specifying the key value. The key value can be used for retrieving data queues by key.

Preparing to Use the Virtual Terminal APIs

The following steps are required to prepare your AS/400 system to run an application using the virtual terminal APIs:

1. Set the number of automatically created virtual terminals using the Automatic virtual device configuration indicator (QAUTOVRT) system value
2. Set the Limit security officer device access (QLMTSECOFR) system value
3. Create user profiles using the Create User Profile (CRTUSRPRF) command

Step 1: Setting the Number of Automatically Created Virtual Terminals

The OS/400 licensed program uses virtual terminals to allow a server program to interact with its client by sending and receiving data with AS/400 applications. The OS/400 operating system will automatically select (and create if necessary) these virtual terminals for you.

The QAUTOVRT system value specifies the maximum number of terminals that will be automatically configured by the system. When you set the QAUTOVRT system value, the OS/400 licensed program automatically configures the required virtual controllers and terminals. Controllers coordinate and control the operation of one or more input/output terminals (such as work stations) and synchronize the operation of such terminals with the operation of the entire system. Use the Change System Value (CHGSYSVAL) command to change the value of the QAUTOVRT system value. For example, entering the following command string changes the number of virtual terminals that can be allocated on a system to 50:

```
CHGSYSVAL SYSVAL(QAUTOVRT) VALUE(50)
```

To determine and set the maximum number of users you want signed on to the AS/400 system at any time, do the following:

- Set the QAUTOVRT system value to 9999, the maximum value allowed.
- Have your users use the AS/400 system until you decide that the number of virtual terminals created is sufficient for normal system operation.
- Use the Work with Configuration Status (WRKCFGSTS) command to determine the number of work stations configured.
- Change the QAUTOVRT system value from 9999 to the number of virtual terminals you require for normal operation.

If you have never allowed virtual terminals to be configured automatically on your system, the QAUTOVRT system value is 0. As a result, you cannot use the virtual terminal APIs because the OS/400 licensed program is not able to create more work stations than the number specified. If you change the QAUTOVRT system value to 10, the next virtual terminal path opened causes the OS/400 licensed program to create a virtual terminal. This virtual terminal is created because the number of virtual terminals on the controller (0) is less than the number specified in the QAUTOVRT system value (10). Even if you change the specified number to 0 again, the next virtual terminal opened may succeed if a virtual terminal exists that is not being used.

If a virtual terminal does not exist or is in use, the OS/400 licensed program does not create a new virtual terminal because the number of virtual terminals currently existing is greater than or equal to the specified QAUTOVRT system value. When the number of virtual terminals that currently exist is greater than or equal to the QAUTOVRT system value, the message CPF8940, "Cannot automatically select virtual device", is sent to the system operator message queue (QSYSOPR). You must either try again when a virtual terminal description becomes available or increase the QAUTOVRT system value.

The OS/400 operating system uses the following conventions for naming virtual controllers and work stations:

- Virtual controllers are named QPACTL*nn*
- Virtual terminal descriptions are named QPADEV*xxxx*

Consider the following when you allow the OS/400 licensed program to automatically configure work stations:

- The OS/400 licensed program does not delete virtual terminals, even when the number of work stations attached to virtual controllers exceeds the limit set by QAUTOVRT.

  If you want the extra work stations deleted, you must manually delete them.

- The OS/400 licensed program allows a maximum of 250 virtual terminals on the QPACTL01 controller before it creates QPACTL02. This value is usually adequate. If you delete work stations to enforce a smaller value for the QAUTOVRT limit, begin by deleting the work stations from the controller with the highest numeric value in its name (where *nn* in the QPACTL*nn* name is largest).

  Note: Changing this system value affects other AS/400 products and programs requiring automatic configuration. This includes TCP/IP TELNET, 5250 display station pass-through, and any other programs using the virtual terminal APIs.

Step 2: Setting the Limit Security Officer (QLMTSECOFR) Value

The Limit security officer device access (QLMTSECOFR) system value, limits the devices the security officer can sign on to. The security officer controls all of the security authorizations provided by the AS/400 system. If the QLMTSECOFR value is greater than zero, the security officer must be authorized to use the virtual device descriptions. However, when this value equals 0, the system does not limit the devices the security officer can use to sign on the system.

When the system security level (QSECURITY) system value is set to 30, a security officer with all object authority (*ALLOBJ) must be authorized to use the work stations. For example, for each display station that a security officer wants to sign on to (local, remote, or virtual), the user must authorize the security officer using the following Grant Object Authority (GRTOBJAUT) command:

```
GRTOBJAUT OBJ(display-name) OBJTYPE(*DEVD) AUT(*CHANGE) USER(QSECOFR)
```

This procedure is very important because using the virtual terminal APIs automatically configures virtual terminals (devices). Automatic configuration is a function that names and creates the descriptions of network devices and controllers attached to a line. If the QLMTSECOFR value is set to 0, all virtual terminals automatically configured when you use the virtual terminal APIs can be used by the security officer. If you set the QLMTSECOFR value to 1, your security officer is not able to use the virtual terminals unless you specifically grant object authority to the security officer for that virtual terminal. The automatic configuration support can delete and re-create the virtual terminal. If this occurs, authority must be granted to the security officer each time the virtual terminal is created.

Security Considerations: The number of sign-on attempts allowed increases if virtual terminals are automatically configured. The number of sign-on attempts is equal to the number of system sign-on attempts allowed multiplied by the number of virtual terminals that can be created. The number of system sign-on attempts allowed is defined by the QMAXSIGN system value. The number of virtual terminals that can be created is defined by the QAUTOVRT system value.

Step 3: Creating User Profiles

You should create one or more user profiles on the AS/400 system for users of the virtual terminal supported by the client and server programs. The default user profile is *SYS. The following example shows a sample user profile:

```
CRTUSRPRF  USRPRF(CLERK1)  PASSWORD(unique-password)
           JOBD(CLERKLIB/CLERKL1)
           TEXT('User profile for one group of clerks')
```

Creating Your Own Virtual Controllers and Devices

You can create your own virtual controllers and devices (terminals); however, you must use the same naming conventions as the automatic controller and device creation support. You may want to create the virtual terminal descriptions to control the number of sign-on attempts possible by not allowing automatic configuration of virtual terminals (which allows additional sign-on attempts to occur). See "Security Considerations" for additional information.

If you do not want to use automatically created descriptions, do the following:

- Use the Create Controller Description (Virtual Work Station) (CRTCTLVWS) command to create a controller description for a virtual terminal.

  ```
  CRTCTLVWS CTLD(QPACTL01)
            TEXT('Virtual Controller for virtual terminals')
  ```

  Note: You must use the OS/400 naming convention, QPACTL*nn*, for naming virtual controllers, where *nn* is a decimal number starting at 01.

- Use the Create Device Description (Display) (CRTDEVDSP) command to create a virtual terminal as follows:

  ```
  CRTDEVDSP DEVD(QPADEV0001) DEVCLS(*VRT)
            TYPE(5251) MODEL(11) CTL(QPACTL01)
            TEXT('24 X 80 Monochrome
               Display for Server Program')
  ```

- The OS/400 licensed program automatically varies on the controller and terminal that you have created. You must use the OS/400 naming convention, QPADEV*nnnn*, for naming virtual device descriptions, where *nnnn* is a decimal number starting at 0001.

After creating the descriptions, you must authorize the server program to use them. Use the Grant Object Authority (GRTOBJAUT) command to authorize the user profile used by the server program to the descriptions. This can be done using the following commands:

```
GRTOBJAUT OBJ(QPACTL01) OBJTYPE(*CTLD)
          AUT(*CHANGE) USER(user-ID)
GRTOBJAUT OBJ(QPADEV0001) OBJTYPE(*DEVD)
          AUT(*CHANGE) USER(user-ID)
```

You may want to prevent virtual terminals from being created automatically. To do this, set the QAUTOVRT system value to 0 as follows:

```
CHGSYSVAL SYSVAL(QAUTOVRT) VALUE(0)
```

See "Step 1: Setting the Number of Automatically Created Virtual Terminals" on page 60-3 for additional information.

Note: Changing this system value affects other AS/400 products and programs requiring automatic configuration. This includes TELNET, 5250 display station pass-through, and any other programs using the virtual terminal APIs.

Developing Client and Server Programs

When developing client and server programs using the virtual terminal APIs, you need to take the following items into consideration:

- The client program should be able to:
  - Interrupt the server program
  - Check the server program's status
  - Discard data from the AS/400 application
- The user should be able to configure a time-out to be used by the client program while waiting for screens from the server program.

- Pressing the Print key on the work station should create a file to be printed at either the work station or the AS/400 printer.

Chapter 61. Virtual Terminal APIs

This chapter describes the syntax and parameters for the virtual terminal APIs. The virtual terminal path APIs consist of the program calls shown in the following list:

Close Virtual Terminal Path (QTVCLOVT) closes one or all open virtual terminal paths.
Open Virtual Terminal Path (QTVOPNVT) opens a path to a virtual terminal.
Read from Virtual Terminal (QTVRDVT) reads data from the virtual terminal into a data buffer.
Send Request for OS/400 Function (QTVSNDRQ) sends a request to perform a particular function.
Write to Virtual Terminal (QTVWRTVT) writes data to a virtual terminal from a data buffer.

The APIs are presented in alphabetical order.

Close Virtual Terminal Path (QTVCLOVT) API

Parameters

Required Parameter Group:

1	Virtual terminal handle	Input	Char(16)
2	Error code	I/O	Char(*)

The Close Virtual Terminal Path (QTVCLOVT) API closes one or all open virtual terminal paths. To close all open virtual terminal paths, the handle must be set to zero.

Required Parameter Group

Virtual terminal handle
INPUT; CHAR(16)
The reference code for the open virtual terminal path, created with the Open Virtual Terminal Path (QTVOPNVT) API. If this parameter is set to zero, all open virtual terminal paths are closed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Open Virtual Terminal Path (QTVOPNVT) API

Parameters

Required Parameter Group:

1	Virtual terminal handle	Output	Char(16)
2	Keyboard language type	Input	Char(3)
3	Character set	Input	Binary(4)
4	Code page	Input	Binary(4)
5	Work station type	Input	Binary(4)
6	Data queue name and library	Input	Char(20)
7	Key value	Input	Char(*)
8	Key value length	Input	Binary(4)
9	Error code	I/O	Char(*)

Optional Parameter:

10	Open operation information	Input	Char(10)

The Open Virtual Terminal Path (QTVOPNVT) API opens a path to a virtual terminal, allowing a server program to interact with an AS/400 application. The virtual terminal path remains open until it is explicitly closed or the job is ended.

When you call the QTVOPNVT API, the operating system selects or automatically configures a virtual terminal for you and indicates that the device is logically turned on. The operating system then creates a sign-on display at the virtual terminal and sends a message to the specified data queue to signal the server program that data is available.

Authorities and Locks

Library Authority *USE
User Queue Authority *CHANGE
User Queue Lock *EXCLRD

Required Parameter Group

Virtual terminal handle
OUTPUT; CHAR(16)
A reference code created by the OS/400 operating system to identify this open virtual terminal path in later calls to other virtual terminal APIs.

Keyboard language type
INPUT; CHAR(3)
The keyboard language type for the virtual terminal. To use the system value, specify blanks for this parameter. For a list of other valid values, see the Create Device Description (CRTDEVDSP) command in the *CL Reference*. For details about supported languages, see the *National Language Support Planning Guide*.

Character set
INPUT; BINARY(4)
The graphic character set for the virtual terminal. Valid values are a specific graphic character set number and these special values:

0 The graphic character set system value is used.
-1 The keyboard language type is used to select the appropriate character set.

For details about the graphic character sets you can specify, see the *National Language Support Planning Guide*.

Note: The graphic character set system value is obtained from the default graphic character set and code page (QCHRID) system value.

Code page
INPUT; BINARY(4)
The code page for the virtual terminal. For details about the code pages you can specify, see the *National Language Support Planning Guide*. If you specified 0 or -1 for the character set parameter, you do not have to specify this parameter. When you use 0 for the character set parameter, the system value is used for this parameter. When you use -1 for the character set parameter, the code page value is derived from the keyboard language type parameter.

Note: The code page system value is obtained from the default graphic character set and code page (QCHRID) system value.

Work station type
INPUT; BINARY(4)
The type of work station to use. Valid values are 1 through 15. See "Supported Work Station Types and Models" for an explanation of the values.

Other work station types and models are supported. You can specify these by determining their equivalents in Figure 61-1. For a more detailed list of work station types and equivalents, see "Supported Work Station Types and Models."

If a virtual terminal description does not yet exist for the work station type specified, the operating system attempts to configure the work station automatically. Automatic configuration is controlled by the Automatic virtual device configuration (QAUTOVRT) system value, which specifies the number of virtual terminals that the operating system can configure automatically. See "Step 1: Setting the Number of Automatically Created Virtual Terminals" on page 60-3 for more information on the QAUTOVRT value.

Data queue name and library
INPUT; CHAR(20)
The name and library of the data queue used by your application program to receive data from the operating system asynchronously. The first 10 bytes are for the data queue name, and the second 10 bytes are for the library name. Allowable special values are:

***CURLIB** The job's current library
***LIBL** The library list

Key value
INPUT; CHAR(*)
The key value to use for the OS/400 data queue.

Key value length
INPUT; BINARY(4)
The length of the key value. Valid values are 0 through 256. If you specify 0, no key is used for data queue entries.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Optional Parameter

Open operation information
INPUT; CHAR(10)
Information about the open operation. The characters and their meanings are:

1 Whether or not the PC text-assist function is supported. Valid values are:

Blank The PC text-assist function is supported.
0 The PC text-assist function is supported.
1 The PC text-assist function is not supported. The adapted word processing function is used automatically.

2-10 Reserved. These characters must be blank.

Supported Work Station Types and Models

Figure 61-1 details the values you can specify for the QTVOPNVT API's work station type parameter:

Figure 61-1 (Page 1 of 2). Work Station Types and Models

Value	Work Station Type and Model	Equivalent Type and Model	Description
1	5251 11		24 x 80 monochrome display.
2	5291 1	5291 2	24 x 80 monochrome display.
3	5292 2		24 x 80 color graphics display. This type is also emulated by a graphics work station feature.

Figure 61-1 (Page 2 of 2). Work Station Types and Models

Value	Work Station Type and Model	Equivalent Type and Model	Description
4	5555 B01	5555 E01	24 x 80 monochrome double-byte character set (DBCS) display. This type is emulated by a monochrome work station feature that supports a DBCS display.
5	3196 A1	3196 A2 3196 B1 3196 B2 3476 EA	24 x 80 monochrome display. This type is emulated by a monochrome work station feature. This is what the ASCII devices emulate.
6	3179 2	3197 C1 3197 C2 3476 EC 5292 1	24 x 80 color display. This type is emulated by a color work station feature.
7	3180 2	3197 D1 3197 D2 3197 W1 3197 W2	27 x 132 monochrome display.
8	3477 FC		27 x 132 wide-screen color display.
9	3477 FG	3477 FA 3477 FD 3477 FE 3477 FW	27 x 132 wide-screen monochrome display.
10	5555 C01	5555 F01	24 x 80 color double-byte character set (DBCS) display. This type is emulated by a color work station feature that supports a DBCS display.
11	5555 G01		24 x 80 double-byte character set (DBCS) monochrome graphics display. This type is emulated by a monochrome work station feature that supports a DBCS display.
12	5555 G02		24 x 80 color double-byte character set (DBCS) graphics display. This type is emulated by a color graphics work station feature that supports a DBCS display.
13	3486 BA		24 x 80 monochrome display.
14	3487 HA	3487 HG 3487 HW	24 x 80 or 27 x 132 monochrome display.
15	3487 HC		24 x 80 or 27 x 132 color display.

Notes:

1. All 5250 work stations, except 5555 Model B01, can operate as 5251 Model 11 work stations.
2. Selecting double-byte character set (DBCS) work stations requires the AS/400 primary language to be one of the DBCS national language versions (NLVs).

Error Messages

CPF1842 E Cannot access system value &1.
CPF3CF1 E Error code parameter not valid.
CPF87FA E Character identifier not valid.
CPF87F0 E Virtual terminal type value &1 not valid.
CPF87F1 E Queue key length &1 not valid.
CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.
CPF87F9 E Keyboard language type &1 not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Read from Virtual Terminal (QTVRDVT) API

Parameters

Required Parameter Group:

1	Virtual terminal handle	Input	Char(16)
2	Read information	Output	Char(10)
3	Data buffer	Output	Char(*)
4	Number of bytes to read	Input	Binary(4)
5	Data received	Output	Binary(4)
6	Error code	I/O	Char(*)

The Read from Virtual Terminal (QTVRDVT) API reads data from the virtual terminal into the server program's data buffer. Your application should read data only if it has received an asynchronous notification message on the data queue, or if the more data flag was set on a previous read operation. The data received is in 5250 data stream format.

Only one full-screen display of data can be received at a time. If the data buffer is too small, partial displays are received and the more data flag for the QTVRDVT API's read information parameter is set to 1.

Before working with 5250 data streams, be sure to see the *IBM 5250 Information Display System Functions Reference Manual.*

Required Parameter Group

Virtual terminal handle
INPUT; CHAR(16)
The reference code for the open virtual terminal path, created by the operating system with the Open Virtual Terminal Path (QTVOPNVT) API.

Read information
OUTPUT; CHAR(10)
Information about the read operation. The characters and their meanings are:

1 The operation code, which gives the server program additional information about OS/400 status and what is expected of the server program. Valid values for this parameter are:

- **1** Invite
- **2** Output only
- **3** Put/get
- **4** Save display
- **5** Restore display
- **6** Read immediate
- **8** Read display
- **A** Cancel invite
- **B** Turn on message light
- **C** Turn off message light

For detailed descriptions of these codes, see "Read Operation Codes."

2 More data flag. Valid values for this parameter are:

- **0** There is no more data.
- **1** More data is available. Issue the read operation again to receive the additional data. This flag is set if the buffer specified on input is not large enough to hold all of the data received from the virtual terminal.

3 Key flag. Valid character values for this parameter are:

- **0** The Enter key was pressed.
- **1** The System Request key was pressed.

4-10 Reserved. These characters must be blank.

Data buffer
OUTPUT; CHAR(*)
The server program's buffer for receiving data from the virtual terminal. The data is a 5250 data stream.

The QTVRDVT API does not lock the data buffer. Thus, other applications should not use the buffer while the API is using it.

The data buffer should be large enough to hold the largest display of data expected. If it is not large enough for all the data to fit, the more data flag of the read information parameter is set to 1. Additional read requests must be performed, until all the remaining data is received and the more data flag is set back to 0.

Number of bytes to read
INPUT; BINARY(4)
The number of bytes to read from the data buffer. This number must be smaller than or equal to the size of the data buffer.

Data received
OUTPUT; BINARY(4)
The amount of data received from the virtual terminal in bytes. If no data is received from the virtual terminal, 0 is returned. Some read operations do not return any data.

For graphic work stations, a maximum of 24 576 (24KB) bytes of data can be returned.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Read Operation Codes

Figure 61-2 describes the operation codes that can be returned on a read request.

Figure 61-2 (Page 1 of 2). Read Operation Codes

Value	Response Name	Description
1	Invite	No data is returned from this read request. The operating system and the application are ready to receive data. The server program is expected to follow this with a write operation when data becomes available from the client program.
2	Output only	This read request returned some data. The server program should send the data to the client program. However, the operating system is not ready to receive data from the server program. The server program should not request any data from the client program yet. This response usually occurs because an application is performing several put operations to the virtual terminal device. After the last put operation by the application, a put/get operation code is usually returned on the read operation.
3	Put/get	Data is returned from this read request and should be sent by the server program to the client program. The operating system is ready to receive data from the server program. The server program should wait for data from the client program.

Figure 61-2 (Page 2 of 2). Read Operation Codes

Value	Response Name	Description
4	Save display	No data is returned from this read request. The operating system expects the server program to obtain the data from the current display and write the data to the virtual terminal. The operating system saves the display for later use, such as returning the display to the server program. The server program must indicate a save-display response on the write operation and send the saved display as data (that is, the saved display must be in the data buffer).
5	Restore display	The data returned is a previously saved display. The server program should send the data to the client program.
6	Read immediate	No data is returned from this read request. The operating system expects the server program to write data to the virtual terminal. Only data from input fields should be written.
8	Read display	No data is returned from this read request. The operating system expects the server program to write data to the virtual terminal. The current display should be written.
A	Cancel invite	No data is returned from this read request. The operating system expects the server program to signal the client program to cancel the outstanding invite operation. When it is canceled, the server program must perform a write operation to the virtual terminal and indicate a cancel invite response. Because the response has no data associated with it, the number of bytes to write must be set to 0 for the write operation.
B	Turn on message light	No data is returned from this read request. A message has been received, and the user should be notified. The server program should signal the client program to turn on a display message indicator light or some other indicator.
C	Turn off message light	No data is returned from this read request. The display message light or some other indicator should be set off.

Error Messages

CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F3 E Data buffer length &1 not valid.
CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

Send Request for OS/400 Function (QTVSNDRQ) API

Parameters

Required Parameter Group

1	Virtual terminal handle	Input	Char(16)
2	Request	Input	Binary(4)
3	Error code	I/O	Char(*)

The Send Request for OS/400 Function (QTVSNDRQ) API sends a request to the operating system to perform a particular function.

Required Parameter Group

Virtual terminal handle
INPUT; CHAR(16)
The reference code for the open virtual terminal path, created with the Open Virtual Terminal Path (QTVOPNVT) API.

Request
INPUT; BINARY(4)
The request to be processed by the operating system. Valid binary values are:

1 Cancel Previous Request: Allows the server program to cancel the previous OS/400 request. This is similar to selecting option 2 on the AS/400 System Request menu.
2 Send Break Message: Causes the operating system to issue a break message to the virtual terminal. You can use this to determine whether the virtual terminal is still active.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F6 E Request value &1 not valid.
CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

Write to Virtual Terminal (QTVWRTVT) API

Parameters

Required Parameter Group:

1	Virtual terminal handle	Input	Char(16)
2	Write information	Input	Char(10)
3	Data buffer	Input	Char(*)
4	Number of bytes to write	Input	Binary(4)
5	Error code	I/O	Char(*)

The Write to Virtual Terminal (QTVWRTVT) API writes data from a server program's data buffer to a virtual terminal. You can send one display to the virtual terminal during each write operation. You cannot send partial or multiple displays.

Required Parameter Group

Virtual terminal handle
INPUT; CHAR(16)
The reference code for the open virtual terminal path, created with the Open Virtual Terminal Path (QTVOPNVT) API.

Write information
INPUT; CHAR(10)
Information about the write operation. The information given in each character is as follows:

1 Key flag. Valid values are:
- **0** The Enter key was pressed.
- **1** The System Request key was pressed. The next read operation returns the AS/400 System Request menu.
- **2** The Attention key was pressed. In this case, the number of bytes to write must be 0.
- **3** The Test Request key was pressed.
- **4** The Help-in-Error key was pressed.

2 Operation code. This parameter describes the type of write operation to perform. Valid values and their meanings are:
- **blank** Put/get
- **2** Output only
- **3** Put/get
- **4** Save display
- **A** Cancel invite

For detailed descriptions of these codes, see "Write Operation Codes."

3 Data stream output error. The negative response code, for example X'10030101', is sent as data in the data buffer.
- **blank** 5250 data in data buffer
- **0** 5250 data in data buffer
- **1** Data stream error; SNA response code data is in the data buffer.

4–10 Reserved. These characters must be blank.

Data buffer
INPUT; CHAR(*)
The server program's buffer containing the data to send to the virtual terminal.

The QTVWRTVT API does not lock the data buffer. Thus, other applications should not use the buffer while the API is using it.

Number of bytes to write
INPUT; BINARY(4)
The number of bytes to write. This number must be smaller than or equal to the size of the data buffer. Valid range of numbers is 0 through 24K. This parameter must be 0 if character 1 of the write information parameter is 2.

Some write operations do not write data.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Write Operation Codes

Figure 61-3 describes the operation codes that can be used for the write information parameter.

Value	Name	Description
blank	Put/get	Data is being sent to the virtual terminal. The virtual terminal server program is ready for input.
2	Output only	This write operation is in response to a read request that returned an output-only read operation code.
3	Put/get	See the description above.
4	Save display	This write operation is in response to a read request that returned a save display read operation code. No data is associated with this write operation; thus, the data buffer length must be set to 0.
A	Cancel invite	This write operation is in response to a read request that returned a cancel invite read operation code. No data is associated with this write operation; thus, the data buffer length must be set to 0.

Figure 61-3. Write Operation Codes

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF87D4 E Data sent exceeded the corresponding I/O request.
CPF87F2 E Virtual terminal handle &1 not valid.
CPF87F3 E Data buffer length &1 not valid.
CPF87F4 E Key flag &1 not valid.
CPF87F5 E Operation code response &1 not valid.

CPF87F7 E Parameter value &1 not valid.
CPF87F8 E Unexpected internal system error occurred in program &1.
| CPF9872 E Program &1 in library &2 ended. Reason code
| &3.

Chapter 62. Virtual Terminal Run-Time Example

To help understand how virtual terminal APIs are used, the following example shows how the OS/400 operating system, server program, client program, and work station device (display and keyboard) interact when processing a system request.

This example starts with the server program waiting for a response from the client program, which is waiting for data from the user (keyboard).

1. System request processing starts when you press the appropriate System Request key on the work station keyboard.
2. The client program informs the server program that the System Request key has been pressed. The protocol used in this case is unique to the particular implementation of these two programs.
3. The server program calls the Write to Virtual Terminal (QTVWRTVT) API for a write request. The flag for the System Request key must be set for this write request. No data is sent to the virtual terminal at this time.
4. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
5. The server program removes the entry from the data queue by calling the Receive Data Queue (QRCVDTAQ) API and then calls the Read from Virtual Terminal (QTVRDVT) API for a read request. The Cancel Invite operation code is returned. No data is received from the virtual terminal at this time.

   To prevent the client program from sending anymore data (screens), the server program informs the client program that it is no longer receiving data from the client program.

   The server program calls the QTVWRTVT API for a write request. The Operation Code parameter is set to Cancel Invite. No data is sent to the virtual terminal at this time.
6. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
7. The server program removes the entry from the data queue by calling the QRCVDTAQ API and then calls the QTVRDVT API for a read request. A Save Screen operation code is returned. No data is received from the virtual terminal at this time.

   The server program gets the current screen. This may require requesting the current screen from the client program.

   The server program calls the QTVWRTVT API for a write request, sending the current screen to the virtual terminal. The Operation Code parameter must be set to Save Screen.
8. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
9. The server program removes the entry from the data queue by calling the QRCVDTAQ API and then calls the QTVRDVT API for a read request. A Put/Get operation code is returned. The data read will be the actual System Request menu.

   The server program sends this data to the client program and waits for a response.
10. The client program updates the display with the System Request menu and waits for a response from the user. The resulting response is sent to the server program.
11. The response is received from the client program, and the server program calls the QTVWRTTV API for a write request, sending the response to the virtual terminal.

    Note: What happens at this point depends on the response to the System Request menu. Additional data may be received from and sent to the virtual terminal. After the response is processed, the following steps occur.
12. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
13. The server program removes the entry from the data queue by calling the QRCVDTAQ API and then performs a call to the QTVRDVT API for a read request. A Put operation code is returned. The data read is the saved (current) screen that was previously written by the server program to the virtual terminal.

    The server program sends the saved screen to the client program but does not wait for a response.
14. The client program updates the work station display with the saved screen.
15. The OS/400 licensed program creates a data queue entry for informing the server program that data is available to be read.
16. The server program removes the entry from the data queue by calling the QRCVDTAQ API. An Invite operation code is returned. Note that no data is received from the virtual terminal at this time.
17. The client program is once again waiting for user data, and the server program is waiting for data from the client program.

Part 24. Work Management APIs

Chapter 63. Work Management APIs

This chapter discusses the AS/400 work management APIs that:

- Retrieve information about subsystems and subsystem job queues, and allow you to change storage pool attributes
- Retrieve information from system flight recorders
- Retrieve information about specific jobs
- Retrieve information about job schedule entries

The work management APIs are:

Change Current Job (QWCCCJOB) changes information for the current job.

Change Pool Attributes (QUSCHGPA) changes the size, activity level, and paging option of system storage pools.

Change Pool Tuning Information (QWCCHGTN) changes information about tuning being performed on the system for the different storage pools.

Control Trace (QWTCTLTR) turns the trace function on and off.

Dump Flight Recorder (QWTDMPFR) dumps the contents of the flight recorders for jobs that have them.

Dump Lock Flight Recorder (QWTDMPLF) dumps the contents of the lock flight recorder for the device that is specified in the parameter that is passed to the program.

List Active Subsystems (QWCLASBS) retrieves a list of active subsystems.

List Job (QUSLJOB) lists some or all jobs on the system.

List Job Schedule Entries (QWCLSCDE) lists the entries in the job schedule QDFTJOBSCD.

List Subsystem Job Queues (QWDLSJBQ) lists the job queues for a subsystem.

Retrieve Data Area (QWCRDTAA) retrieves the contents of a data area.

Retrieve Job Description Information (QWDRJOBD) retrieves information from a job description object.

Retrieve Job Information (QUSRJOBI) retrieves information, such as job attributes and performance data about a specific job.

Retrieve Job Queue Information (QSPRJOBQ) retrieves information associated with a specified job queue.

Retrieve Network Attributes (QWCRNETA) retrieves network attributes.

Retrieve Subsystem Information (QWDRSBSD) retrieves information about a specific subsystem.

Retrieve System Status (QWCRSSTS) retrieves a group of statistics that represent the current status of the system.

Retrieve System Values (QWCRSVAL) retrieves system values.

Set Lock Flight Recorder (QWTSETLF) turns the lock flight recorder on and off.

Set Trace (QWTSETTR) starts the Trace Job (TRCJOB) command for the job passed on the job and user name parameter at the earliest point while the job is starting.

The APIs in this chapter are presented in alphabetical order.

Change Current Job (QWCCCJOB) API

Parameters

Required Parameter Group:

1	Changed job information	Input	Char(*)
2	Error code	I/O	Char(*)

The Change Current Job (QWCCCJOB) API lets you change information for the current job. The user can change the Cancel or the Exit keys.

Required Parameter Group

Changed job information
INPUT; CHAR(*)
The information for the job that you want to change. The information must be in the following format:

Number of variable length records
BINARY(4)
Total number of all of the variable length records.

Variable length records
The fields of the job to change and the data used for the change. For the specific format of the variable length record, see the "Format for Variable Length Record."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format for Variable Length Record

The following table defines the format for the variable length records.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Key identifier
4	4	BINARY(4)	Length of key data
8	8	CHAR(*)	Key data

If the length of the data is longer than the key identifier's data length, the data will be truncated at the right. No message will be issued.

If the length of the data is smaller than the key identifier's data length, the data will be padded with blanks at the right. No message will be issued.

Field Descriptions

Key data. The data used to change a specific field of the job.

Key identifier. The field of the job to change. Only specific fields of the job can be changed. See "Key Identifiers" for the list of valid keys.

Length of key data. The length of the data used to change a specific field of the job.

Key Identifiers: The following table lists the valid keys for the key identifier area of the variable length record.

Key ID	Type	Field description
1	CHAR(1)	Exit key
2	CHAR(1)	Cancel key

Key Identifier Descriptions

Exit key. Whether or not the Exit key is set as pressed for the job. It must have a value of 0 or 1.

0 The Exit key was not pressed.
1 The Exit key was pressed.

Note: The application or command that was called before this API determines how the key is set.

Cancel key. Whether or not the Cancel key is set as pressed for the job. It must have a value of 0 or 1.

0 The Cancel key was not pressed.
1 The Cancel key was pressed.

Note: The application or command that was called before this API determines how the key is set.

Error Messages

CPF1863 E Length of data not valid.
CPF1867 E Value for key is not valid.
CPF1868 E Number of records is not valid.
CPF2199 E &2 not valid for field &1.
CPF24B4 E Severe error addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Change Pool Attributes (QUSCHGPA) API

Parameters

Required Parameter Group:

1	System pool identifier	Input	Binary(4)
2	New pool size	Input	Binary(4)
3	New pool activity level	Input	Binary(4)

Optional Parameter Group:

4	Message logging	Input	Char(1)
5	Error code	I/O	Char(*)

Optional Parameter:

6	Paging option	Input	Char(10)

The Change Pool Attributes (QUSCHGPA) API changes the size, activity level, and paging options of system storage pools. A system storage pool identifier is returned with the Materialize Resource Management Data (MATRMD) machine interface (MI) instruction. (Note that *system* pool identifiers differ from *subsystem* pool identifiers.) Depending on whether the base pool, shared pool, or private subsystem pool is to be changed, the QUSCHGPA API determines the appropriate command to use and then issues that command. This is similar to the function provided on the System Status display, where you can change the system storage pool size and paging options interactively.

You can use the QUSCHGPA API to tune storage pools (without having to know which subsystem monitor allocated the pool and without having to determine whether or not a pool is a shared storage pool). The Work with System Status (WRKSYSSTS) and the Work with Subsystems (WRKSBS) commands combined provide similar functions.

Authorities and Locks

Subsystem Description Authority *OBJOPR
Description Library Authority *OBJOPR

Required Parameter Group

System pool identifier
INPUT; BINARY(4)
This identifies which pool is to be changed. This number corresponds to the number returned on option nine of the MATRMD MI instruction using the System C/400 PRPQ. This also corresponds to the identifier shown on the Work with System Status display. This parameter is a value ranging from 1 to 16 where pool 1 is the machine pool, and pool 2 is the base pool.

New pool size
INPUT; BINARY(4)
The size of the pool in kilobytes, where one kilobyte is 1024 bytes. If you do not want the pool size to be changed, you must specify a value of -1 for this parameter. The minimum value is 32 kilobytes.

New pool activity level
INPUT; BINARY(4)
The activity level for the pool. If you do not want the activity level to be changed, you must specify a value of -1 for this parameter. You cannot change the activity level of the machine pool.

Optional Parameter Group

Message logging
INPUT; CHAR(1)
Whether messages reporting that a change was made are written to the current job's job log and to the QHST message log. This affects the logging of change-related messages only; it does not affect the logging of error messages. Valid values are:

Y Log change messages.
N Do not log change messages.

If this parameter is omitted, Y is used and change messages are logged.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Optional Parameter

Paging option
INPUT; CHAR(10)
Whether the system should dynamically adjust the paging characteristics of the storage pool for optimum performance. Valid values are:

***SAME** The paging option for the storage pool is not changed.
***FIXED** The system will not dynamically adjust the paging characteristics; system default values are used.
***CALC** The system will dynamically adjust the paging characteristics.

If this parameter is omitted, the paging option is not changed.

The following table summarizes the values you can specify for the system pool identifier, the new pool size, and the new pool activity level.

System Pool Identifier	New Pool Size	New Pool Activity Level
1 Machine pool	-1 or ≥ 256	-1
2 Base pool	-1 or ≥ 32	-1 or 1 through 32 767
3 to 16	≥ 1	1 through 32 767

For pools 3-16, both size and pool activity level must be specified.

In some cases, pool size changes do not take effect immediately. For example, a save or restore operation might be using some of the storage allocated to a pool, or the system might be using some of the storage allocated to the base pool. The size is changed only when the storage being used is free again.

The base pool holds all unused main storage on the system that is not allocated to other shared or private pools. As subsystems are started and allocate storage for their shared and private storage pools, that storage comes from the base pool. The base pool (pool number 2) size is what is left after pool 1 and pools 3 through 16 are subtracted from the total main storage. The QBASPOOL system value is a minimum size, and it is not the actual size of the base pool. At this minimum size, the system does not allow additional storage requests. For this reason, you must calculate the storage requirements for all pools on the system, including the base pool, and then run this API.

Error Messages

CPF1001 E Wait time ends for system response.
CPF1076 E Specified value not allowed for system value &1.
CPF1078 E System value &1 not changed.
CPF1619 E Subsystem description &1 in library &2 damaged.
CPF1691 E Active subsystem description may or may not have changed.
CPF1697 E Subsystem description &1 not changed.
CPF1879 E Paging option &1. not valid.
CPF1880 E Machine pool paging options cannot be changed.
CPF1881 E Changing private pool paging option not allowed.
CPF24B4 E Severe error while addressing parameter list.
CPF3CA0 E System pool &1 does not exist.
CPF3CA1 E Pool size &1 is not valid.
CPF3CA2 E Activity level &1 is not valid.
CPF3CA3 E Pool &1 is not in use.
CPF3CA4 E Changing machine pool activity level is not allowed.
CPF3CA5 E Both pool size and activity level are required.
CPF3CA6 E Message logging value &1 not valid.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Example: Changing System Storage Pool Attributes

The following is an example of how to change system storage pool attributes using the QUSCHGPA API:

```
Pseudocode
     .
     .
     .
MATRMD (OPERAND1, OPERAND2);
DO             /* Do loop for each pool in use    */
     .
     .         /* Calculate the desired pool size
                  and activity level              */
     .
END
DO             /* Do loop for each pool in use    */
CALL QUSCHGPA (POOLID, POOLSIZE, POOLACTLVL);
               /* Change pool attributes          */
END
     .
     .
     .
```

Change Pool Tuning Information (QWCCHGTN) API

Parameters

Required Parameter Group:

1	System pool identifier	Input	Binary(4)
2	Change request	Input	Char(*)
3	Length of change request	Input	Binary(4)
4	Format name	Input	Char(8)
5	Error code	I/O	Char(*)

The Change Pool Tuning Information (QWCCHGTN) API changes information about tuning being performed by the system for the different storage pools. The Materialize Resource Management Data (MATRMD) machine interface (MI) instruction can be used to retrieve the current setting of the tuning parameters.

Required Parameter Group

System pool identifier
INPUT; BINARY(4)
The pool is to be changed. This number corresponds to the number returned on option 9 of the Materialize Resource Management Data (MATRMD) MI instruction. This also corresponds to the identifier shown on the Work with System Status display. This parameter is a value ranging from 2 to 16, where pool 2 is the base pool.

Change request
INPUT; CHAR(*)
The variable containing the new tuning information. See "TUNI0100 Format" for the definition of the fields for this parameter.

Length of change request
INPUT; BINARY(4)
The length of the change request list. This area must be as large as the format specified.

Format name
INPUT; CHAR(8)
The format of the information to be changed. The valid values are:

TUNI0100 Tuning information for a storage pool.

Error code
I/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.

TUNI0100 Format

The following table shows the information that must be specified in the change request parameter when format TUNI0100 is specified. For a detailed description of each field, see "Field Descriptions" on page 63-5.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(1)	Type of tuning
1	1	CHAR(1)	Change page handling
2	2	BINARY(4)	Blocking factor for nondatabase objects
6	6	CHAR(1)	Allow exchange operations (for class 1 objects in pool)
7	7	CHAR(1)	Type of transfer from main storage to auxiliary storage (for class 1 objects in pool)
8	8	BINARY(4)	Blocking factor for database (class 1) objects
12	C	CHAR(1)	Allow exchange operations (for class 2 objects in pool)
13	D	CHAR(1)	Type of transfer from main storage to auxiliary storage (for class 2 objects in pool)
14	E	BINARY(4)	Blocking factor for database (class 2) objects
18	12	CHAR(1)	Allow exchange operations (for class 3 objects in pool)
19	13	CHAR(1)	Type of transfer from main storage to auxiliary storage (for class 3 objects in pool)
20	14	BINARY(4)	Blocking factor for database (class 3) objects
24	18	CHAR(1)	Allow exchange operations (for class 4 objects in pool)

Offset		Type	Field
Dec	Hex		
25	19	CHAR(1)	Type of transfer from main storage (for class 4 objects in pool) to auxiliary storage
26	1A	BINARY(4)	Blocking factor for database (class 4) objects

When tuning is requested (values 1, 2, or 3 for the type of tuning field), the system periodically categorizes database objects into four different performance classes. The classes are:

Class 1 Object access appears to be random. A disk access is required for nearly each record that is accessed.

Class 2 Locality of reference detected. Several records are being accessed per disk access.

Class 3 High locality of reference detected. The object is being processed in a sequential manner; references are highly clustered and large portions of the object are resident in main storage.

Class 4 The class of a database object is adjusted if the object's size is small in comparison to the available storage in the storage pool. This class adjustment involves adding 1 to the class number; therefore, a class 3 database object (as defined above) would be treated as a class 4 if it were small in comparison to the available storage in the storage pool.

Reference information for determining an object's class is collected periodically. It is collected by storage pool because an object's class varies over time and by storage pool.

Note: When a new system pool is created as a result of starting a subsystem, the type of tuning and change page handling attributes for the new system pool are initialized based on the type of storage pool being created. For shared storage pools, the type of tuning and change page handling attributes are set based on the paging option defined for the shared storage pool. For private storage pools, the type of tuning attribute is set to indicate no tuning should be done and the change page handling attribute is set to the system default value.

Field Descriptions

Allow exchange operations. The exchange operation used to reduce the working set size. This is done by overlaying data that is already in main storage with new data this is being brought into main storage. The values for this field are:

0 Use the system default
1 Allow exchange operations
2 Disable exchange operations
3 Disable exchange operations (The data that already exists in main storage should be a good candidate to be replaced when additional storage is needed in the storage pool.)

The value specified for this field is ignored unless static tuning is specified for the type of tuning field.

Blocking factor for database objects. The amount of data that should be brought into main storage when a request is made to read database objects from auxiliary storage. The values for this field are:

0 Use the system default
4 Transfer data into main storage in 4KB blocks
8 Transfer data into main storage in 8KB blocks
16 Transfer data into main storage in 16KB blocks
32 Transfer data into main storage in 32KB blocks
64 Transfer data into main storage in 64KB blocks
128 Transfer data into main storage in 128KB blocks

The system may need to issue multiple I/O operations to bring the data into main storage. The value specified for the blocking factor for database objects field is ignored unless static tuning is specified for the type of tuning field.

Blocking factor for nondatabase objects. The amount of data that should be brought into main storage when a request is made to read nondatabase objects from auxiliary storage. The possible values for this field are:

0 Use the system default
4 Transfer data into main storage in 4KB blocks
8 Transfer data into main storage in 8KB blocks
16 Transfer data into main storage in 16KB blocks
32 Transfer data into main storage in 32KB blocks

The system may need to issue multiple I/O operations to bring the data into main storage. The value specified for the blocking factor for nondatabase objects is ignored unless static tuning is specified for the type of tuning field.

Change page handling. The method the system uses to determine when to write changed pages to auxiliary storage. The values for this field are:

0 Use the system default
1 Changed pages should be written to auxiliary storage when there is a demand for pages in a storage pool
2 In addition to writing changed pages on demand, periodically write changed pages to auxiliary storage

Type of transfer from main storage to auxiliary storage. The method the system uses to process a request to write an object to auxiliary storage. The values for this field are:

0 Use the system default.
1 When objects are changed, write the changes to auxiliary storage. Indicate that the portion of the object that was written to auxiliary storage should be a good candidate to be replaced when additional storage is needed in the storage pool.
2 When objects are changed, write the changes to auxiliary storage.

3 Do not immediately write the changes to auxiliary storage. Indicate that the portion of the object that was changed should be a good candidate to be replaced when additional storage is needed in the storage pool.

4 Do not immediately write the changes to auxiliary storage.

The value specified for this field is ignored unless static tuning is specified for the type of tuning field.

Type of tuning. The method used by the system to tune the storage pool. The values for this field are:

0 No tuning is being performed for this pool.

All values specified for the blocking factor, the allow exchange operations, and the type of transfer from main storage to auxiliary storage fields are ignored. The system default values are used for all these fields.

1 Static tuning is being performed for this pool. Static tuning implies that the values specified for blocking factor, exchange operation, and transfers to auxiliary storage are not dynamically adjusted by the system.

Values must be specified for the blocking factor, allow exchange operations, and type of transfer from main storage to auxiliary storage for the storage pools.

2 Dynamic tuning of transfers into main storage is being performed. This indicates that the system is dynamically adjusting the blocking factor and exchange operations.

Because the values for blocking factor and allow exchange operations are dynamically adjusted, the values specified on the API are ignored. The value used for the transfer to auxiliary storage field is set to ensure that requests to write data to auxiliary storage are processed immediately.

3 Dynamic tuning of transfers into main storage and to auxiliary storage is being performed. This indicates that the system is dynamically adjusting the blocking factor, exchange operations, and transfers to auxiliary storage.

Because the values for the blocking factor, allow exchange operations, and transfers to auxiliary storage are dynamically adjusted, the values specified on the API are ignored.

Error Messages

CPF1001 E Wait time ends for system response.
CPF1870 E Value &1 for type of tuning not valid.
CPF1871 E Value &1 for change page handling not valid.
CPF1872 E Value &1 for blocking factor not valid.
CPF1873 E Value &1 for exchange operation not valid.
CPF1874 E Value &1 for transfer to auxiliary storage not valid.
CPF1875 E Value &1 for change request length not valid.
CPF1876 E Value &1 for pool number not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error occurred during running of &1 API.
CPF3C21 E Format name &1 not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Control Trace (QWTCTLTR) API

Parameters

Required Parameter:			
1	Control value	Input	Char(10)
Optional Parameter:			
2	Error code	I/O	Char(*)

The Control Trace (QWTCTLTR) API turns the early trace function on and off. The value of *ON is passed to the program to turn the early tracing on, and *OFF is passed to turn the early tracing off.

When the early trace function is turned on, the jobs that are set up by the Set Trace (QWTSETTR) API begin tracing as soon as they are started. The tracing is stopped by turning it off with the Control Trace (QWTCTLTR) API. When *OFF or *RESET is passed to the program, this causes the trace information for the jobs to dump to spooled files.

The information set up by this API remains in effect during an initial program load (IPL).

This API should be used only when recommended by your IBM service representative.

Required Parameter

Control value
INPUT; CHAR(10)
The value passed to turn the early trace function on or off. The valid values are:

***ON** Turns early tracing on.
***OFF** Turns early tracing off and stops any trace activity started by this API.
***RESET** Turns early tracing off, stops any trace activity started by this API, and clears the space that contains the information that was set up by the QWTSETTR API.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

| CPF119D E Value &1 specified for parameter is not valid.
| CPF24B4 E Severe error while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Dump Flight Recorder (QWTDMPFR) API

The Dump Flight Recorder (QWTDMPFR) API dumps the contents of flight recorders for jobs that have them. A **flight recorder** is an object that stores trace information to record a history of what has happened in system programs. The flight recorder contains only information that helps to identify the flow of system programs and status information.

The following types of jobs have flight recorders:

- Subsystem monitors
- System jobs

The QWTDMPFR API has no parameters.

You can use the QWTDMPFR API to collect information for your IBM service representative. This API dumps the contents of job flight recorders to a spooled file. You can then collect the files and submit them to your IBM service representative for debugging.

Dump Lock Flight Recorder (QWTDMPLF) API

Parameters

Required Parameter:

1	Device name	Input	Char(10)

Optional Parameter:

2	Error code	I/O	Char(*)

The Dump Lock Flight Recorder (QWTDMPLF) API dumps the following information into spooled files:

- The contents of the lock flight recorder for the device specified in the parameter passed to the program
- QSYSARB job log
- QLUS job log
- Job logs of the active jobs that have used the device as indicated in the lock flight recorder data
- The history log (QHST)
- Device description of the device
- Controller description of the controller to which the device is attached
- Line description of the line to which the controller is attached
- A Work with Object Locks (WRKOBJLCK) listing for the device
- A Work with Configuration Status (WRKCFGSTS) listing for the controller
- The subsystem description of active subsystems that have touched the device
- Associated internal system objects

You can use the QWTDMPLF API to collect information for your IBM service representative.

Required Parameter

Device name
INPUT; CHAR(10)
The name of the device for which flight recorder information will be dumped.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

| CPF119C E Value &1 specified for parameter is not valid.
| CPF3CF1 E Error code parameter not valid.
| CPF8100 E All CPF81xx messages could be signalled. xx is from 01 to FF.
| CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Active Subsystems (QWCLASBS) API

Parameters

Required Parameter Group:

1	Receiving user space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	Error code	I/O	Char(*)

The List Active Subsystems (QWCLASBS) API retrieves a list of active subsystems. QWCLASBS replaces any subsystem list that already exists in the user space. It does not add the new list to an existing one. To retrieve more information about active subsystems, see "Retrieve Subsystem Information (QWDRSBSD) API" on page 63-40.

Authorities and Locks

User Space Authority *CHANGE
Library Authority *USE

User Space Lock *EXCLRD

Required Parameter Group

Receiving user space and library name
INPUT; CHAR(20)
The user space that receives the list, and the library in which it is located. The first 10 characters contain the user space name. The second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The format to use for the list of active subsystems. You can use this format name:

SBSL0100 Basic subsystem list. See "Format of the Generated List."

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated List

The list of active subsystems that the QWCLASBS API returns into the user space consists of:

- A user area
- A generic header
- An input parameter section
- A list data section

The user area and generic header are described in "User Space Format for List APIs" on page 2-7. The remaining items are described in the following sections. For detailed descriptions of the fields in the tables, see "Field Descriptions."

Input Parameter Section

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library specified

SBSL0100 Format: This section is repeated for each active subsystem.

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Subsystem description name
10	A	CHAR(10)	Subsystem description library name

Field Descriptions

Subsystem description library name. The name of the library in which the active subsystem description resides.

Subsystem description name. The name of the active subsystem about which information is being returned.

User space library specified. The library name or special value specified in the call to this API.

User space name. The name of the user space that receives the list.

Error Messages

| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF8122 E &8 damage on library &4.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9838 E User profile storage limit exceeded.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Job (QUSLJOB) API

Parameters

Required Parameter Group:

1	Receiving user space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	Qualified job name	Input	Char(26)
4	Status	Input	Char(10)

| Optional Parameter Group1:

5	Error code	I/O	Char(*)

| Optional Parameter Group 2:

6	Job type	Input	Char(1)
7	Number of fields to return	Input	Binary(4)
8	Key of fields to return	Input	Array(*) of Binary(4)

The List Job (QUSLJOB) API generates a list of all or some jobs on the system. The generated list replaces any existing list in the user space.

The QUSLJOB API produces a list similar to the list produced by the Work with User Job (WRKUSRJOB) command.

Authorities and Locks

User Space Authority *CHANGE
Library Authority *USE
User Space Lock *EXCLRD
Job Authority *JOBCTL, if the job for which information is retrieved has a different user profile from that of the job that calls the QUSLJOB API. *JOBCTL is only needed if format JOBL0200 is specified.

Required Parameter Group

Receiving user space and library name
INPUT; CHAR(20)
The user space that is to receive the generated list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Format name
INPUT; CHAR(8)
The format of the job list to be returned. If format JOBL0200 is specified, the fields that were selected by the caller will be returned for each job in the list. This format is slower than the JOBL0100 format. The performance will vary depending on the number of fields selected.

You must use one of the following format names:

JOBL0100 Basic job list.
JOBL0200 Basic job list with keyed return fields.

For more information, see "Format of the Generated List" on page 63-10.

Qualified job name
INPUT; CHAR(26)
The name of the job to be included in the list. The qualified job name has three parts:

Job name
CHAR(10). A specific job name, a generic name, or one of the following special values:
***** Only the job that this program is running in. The rest of the qualified job name parameter must be blank.
***CURRENT** All jobs with the current job's name.
***ALL** All jobs. The rest of the job name parameter must be specified.

User name
CHAR(10). A specific user profile name, a generic name, or one of the following special values:
***CURRENT** Jobs with the current job's user profile.
***ALL** Jobs with the specified job name, regardless of the user name. The rest of the job name parameter must be specified.

Job number
CHAR(6). A specific job number or the following special value:
***ALL** Jobs with the specified job name and user name, regardless of the job number. The rest of the job name parameter must be specified.

Status
INPUT; CHAR(10)
The status of the jobs to be included in the list. The special values supported are:

***ACTIVE** Active jobs. This includes group jobs, system request jobs, and disconnected jobs.
***JOBQ** Jobs currently on job queues.
***OUTQ** Jobs that have completed running but still have output on an output queue.
***ALL** All jobs, regardless of status.

Optional Parameter Group 1

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Optional Parameter Group 2

Job type
INPUT; CHAR(1)
The type of job to be listed. Refer to "Comparing Job Type and Subtype with the Work with Active Job Command" on page 63-34 for information about how the job type field and the job subtype field equate to the type field in the Work with Active Job (WKRACTJOB) command.

The possible values for this parameter are:

***** This value lists all job types.
A The job is an autostart job.
B The job is a batch job.
I The job is an interactive job.
M The job is a subsystem monitor job.
R The job is a spooled reader job.
S The job is a system job.

W The job is a spooled writer job.
X The job is the start-control-program-function (SCPF) system job.

Number of fields to return
INPUT; BINARY(4)
The number of fields to return in the JOBL0200 format. This parameter is only used for the JOBL0200 format. If JOBL0100 is specified for the format name, the value must be zero.

Key of fields to be returned
INPUT; ARRAY(*) of BINARY(4)
The list of the fields to be returned in the JOBL0200 format. For a list of the valid fields, see "Valid Keys" on page 63-12. This parameter is only used for the JOBL0200 format. If JOBL0100 is specified for the format name, the value must be zero.

Format of the Generated List

The job list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

For details about the user area and generic header, see "User Space Format for List APIs" on page 2-7. For details about the remaining items, see the following sections. For detailed descriptions of the fields in the list returned, see "Field Descriptions."

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." For an example of the command source for the Delete Old Spooled Files (DLTOLDSPLF) command, see "DLTOLDSPLF Command Source" on page A-1.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job name specified
10	A	CHAR(10)	User name specified
20	14	CHAR(6)	Job number specified
26	1A	CHAR(10)	Status
36	24	CHAR(10)	User space specified
46	2E	CHAR(10)	User space library specified
56	38	CHAR(8)	Format name specified
64	40	CHAR(1)	Job type specified
65	41	CHAR(3)	Reserved
68	44	BINARY(4)	Number of fields to return specified
72	48	ARRAY(*) of BINARY(4)	Key of fields to return specified

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job name used
10	A	CHAR(10)	User name used
20	14	CHAR(6)	Job number used

JOBL0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job name used
10	A	CHAR(10)	User name used
20	14	CHAR(6)	Job number used
26	1A	CHAR(16)	Internal job identifier
42	2A	CHAR(10)	Status
52	34	CHAR(1)	Job type
53	35	CHAR(1)	Job subtype
54	36	CHAR(2)	Reserved

JOBL0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(56)	Everything in JOBL0100 format
56	38	CHAR(1)	Job information status
57	39	CHAR(3)	Reserved
60	3C	BINARY(4)	Number of fields returned
These fields repeat, in the order listed, for each key selected.		BINARY(4)	Length of field information returned
		BINARY(4)	Key field
		CHAR(1)	Type of data
		CHAR(3)	Reserved
		BINARY(4)	Length of data
		CHAR(*)	Data
		CHAR(*)	Reserved

Field Descriptions

Data. The data returned for the key field.

Format name specified. The format name as specified in the call to the API.

Internal job identifier. A value sent to other APIs to speed the process of locating the job on the system. Only APIs described in this manual use this identifier. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Job information status. Whether the information was available for the job. The possible values are:

blank The information was available.
A The user was not authorized to the job.
L The information was not available because the job was not accessible.

Job name specified. The name of the job as specified in the call to the API.

Job name used. The name of the job as identified to the system. For an interactive job, the system assigns the job the name of the work station where the job started; for a batch job, you specify the name in the command when you submit the job.

Job number specified. The job number as specified in the call to the API.

Job number used. The system-assigned job number.

Job subtype. Additional information about the job type (if any exists). Refer to "Comparing Job Type and Subtype with the Work with Active Job Command" on page 63-34 for information about how the job type field and the job subtype field equate to the type field in the Work with Active Job (WKRACTJOB) command. The possible values are:

blank The job has no special subtype.
D The job is a batch immediate job.
E The job started with a procedure start request.
J The job is a prestart job.
P The job is a print driver job.
T The job is a System/36 multiple requester terminal (MRT) job.
U Alternate spool user.

Job type. The type of job. Refer to "Comparing Job Type and Subtype with the Work with Active Job Command" on page 63-34 for information about how the job type field and the job subtype field equate to the type field in the Work with Active Job (WKRACTJOB) command. The possible values for this field are:

A The job is an autostart job.
B The job is a batch job.
I The job is an interactive job.
M The job is a subsystem monitor job.
R The job is a spooled reader job.
S The job is a system job.
W The job is a spooled writer job.
X The job is the SCPF system job.

Job type specified. The job type as specified in the call to the API.

Key field. The field returned. See "Valid Keys" on page 63-12 for the list of valid keys.

Key of fields to return specified. The key of fields to return as specified in the call to the API.

Length of data. The length of the data returned for the field.

Length of field information returned. The total length of information returned for this field. This value is used to increment to the next field in the list.

Number of fields returned. The number of fields returned to the application.

Number of fields to return specified. The number of fields to return as specified in the call to the API.

Reserved. An ignored field.

Status. The status of the job. The valid values are:

**ACTIVE* The job has started, and it can use system resources (processing unit, main storage, and so on). This does not guarantee that the job is currently running, however. For example, an active job may be in one of the following states where it is not in a position to use system resources:

- The Hold Job (HLDJOB) command holds the job; the Release the (RLSJOB) command allows the job to run again.
- The Transfer Group Job (TFRGRPJOB) or Transfer Secondary Job (TFRSECJOB) command suspends the job. When control returns to the job, the job can run again.
- The job is disconnected using the Disconnect Job (DSCJOB) command. When the interactive user signs back on, thereby connecting back into the job, the job can run again.
- The job is waiting for any reason. For example, with an inquiry message when the job receives the reply, the job can start running again.

**JOBQ* The job is currently on a job queue. The job possibly was previously active and was placed back on the job queue because of the Transfer Job (TFRJOB) or Transfer Batch Job (TFRBCHJOB) command, or the job was never active because it was just submitted.

**OUTQ* The job has completed running and has spooled output that has not yet printed.

Type of data. The type of data returned.

C The data is returned in character format.
B The data is returned in binary format.

User name specified. The user name as specified in the call to the API.

User name used. The user profile under which the job is run. The user name is the same as the user profile name and can come from several different sources depending on the type of job.

User space library specified. The name of the library containing the user space as specified in the call to the API.

User space specified. The name of the user space as specified in the call to the API.

Valid Keys

The following table contains a list of the valid keys.

See "Field Descriptions" on page 63-28 for the descriptions of the valid key fields.

Key	Type	Description
0101	CHAR(4)	Active job status
0201	CHAR(10)	Break message handling
0301	CHAR(1)	Cancel key
0302	BINARY(4)	Coded character set ID
0303	CHAR(2)	Country ID
0304	BINARY(4)	Processing unit used
0305	CHAR(10)	Current user profile
0306	CHAR(1)	Completion status
0401	CHAR(13)	Date and time job became active
0402	CHAR(13)	Date and time job entered system
0403	CHAR(8)	Date and time job is scheduled to run
0404	CHAR(8)	Date and time job was put on this job queue
0405	CHAR(4)	Date format
0406	CHAR(1)	Date separator
0407	CHAR(1)	DBCS-capable
0408	CHAR(10)	DDM conversation handling
0409	BINARY(4)	Default wait
0410	CHAR(13)	Device recovery action
0501	BINARY(4)	End severity
0502	CHAR(1)	End status
0503	CHAR(1)	Exit key
0601	CHAR(10)	Function name
0602	CHAR(1)	Function type
0701	CHAR(1)	Signed-on job
0901	CHAR(10)	Inquiry message reply
1001	CHAR(15)	Job accounting code
1002	CHAR(7)	Job date
1003	CHAR(20)	Job description name
1004	CHAR(20)	Job queue name
1005	CHAR(2)	Job queue priority
1006	CHAR(8)	Job switches
1201	CHAR(3)	Language ID
1202	CHAR(1)	Logging level
1203	CHAR(10)	Logging of CL programs
1204	BINARY(4)	Logging severity
1205	CHAR(10)	Logging text
1301	CHAR(8)	Mode name
1302	BINARY(4)	Maximum processing unit time
1303	BINARY(4)	Maximum temporary storage
1401	BINARY(4)	Number of auxiliary I/O requests
1402	BINARY(4)	Number of interactive transactions
1501	CHAR(20)	Output queue name
1502	CHAR(2)	Output queue priority
1601	CHAR(10)	Print key format
1602	CHAR(30)	Print text
1603	CHAR(10)	Printer device name
1604	CHAR(10)	Purge
1605	BINARY(4)	Product return code
1606	BINARY(4)	Program return code
1801	BINARY(4)	Response time total
1802	BINARY(4)	Run priority
1901	CHAR(20)	Sort sequence
1902	CHAR(10)	Status message handling
1903	CHAR(10)	Status of the job on the job queue
1904	CHAR(26)	Submitter's job name
1905	CHAR(20)	Submitter's message queue name
1906	CHAR(20)	Subsystem description name
1907	BINARY(4)	System pool identifier
1908	CHAR(10)	Special environment
2001	CHAR(1)	Time separator
2002	BINARY(4)	Time slice
2003	CHAR(10)	Time-slice end pool
2004	BINARY(4)	Temporary storage used
2101	CHAR(24)	Unit of work ID
2102	BINARY(4)	User return code

Error Messages

CPF1865 E Value &1 for job type is not valid.
CPF1866 E Number of fields to return is not valid.
CPF1867 E Value &1 for key value is not valid.

| CPF24B4 E Severe error while addressing parameter list.
| CPF3CB1 E Value &1 for job status is not valid.
| CPF3CB2 E Value specified for job parameter is not valid.
| CPF3CF1 E Error code parameter not valid.
| CPF3C20 E Error found by program &1.
| 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 signalled. xx is from 01 to FF.
| CPF9800 E All CPF98xx messages could be signalled. xx is from 01 to FF.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 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.

List Job Schedule Entries (QWCLSCDE) API

Parameters

Required Parameter Group:

1	User space and library name	Input	Char(20)
2	Format name	Input	Char(8)
3	Job schedule entry name	Input	Char(10)
4	Continuation handle	Input	Char(16)
5	Error code	I/O	Char(*)

The List Job Schedule Entries (QWCLSCDE) API lists the entries in the job schedule, QDFTJOBSCD. A subset of the list can be created by using the job schedule entry name parameter. The generated list replaces any existing list in the user space.

The QWCLSCDE API produces a list similar to the list produced by the Work with Job Schedule Entries (WRKJOBSCDE) command.

Authorities and Locks

User Space Authority *CHANGE
User Space Library Authority *USE
User Space Lock *EXCLRD
Job Schedule Entry Authority *USE if using format SCDL0100; *JOBCTL or the adder of the entry if using format SCDL0200
Job Schedule Authority *USE
Job Schedule Library Authority *READ
Job Schedule Lock *SHRRD

Required Parameter Group

User space and library name
INPUT; CHAR(20)
The user space that is to receive the created list. The first 10 characters contain the user space name, and the second 10 characters contain the name of the library where the user space is located. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

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

SCDL0100 Basic job schedule entries list.
SCDL0200 Detailed job schedule entries list. This format requires more processing than the SCDL0100 format.

For more information, see "SCDL0100 Format" on page 63-14 or "SCDL0200 Format" on page 63-14.

Job schedule entry name
INPUT; CHAR(10)
The job schedule entry about which to retrieve information. This can be used to create a subset of job schedule entries by using the following values:

***ALL** All of the job schedule entries are returned in the list.
generic* All of the job schedule entries beginning with the generic value are returned in the list.
name The job schedule entries with the given name are returned in the list.

Continuation handle
INPUT; CHAR(16)
The value returned to the user in the header section when a partial list is returned. The possible values are:

blank This will start at the beginning of the list.
value The entries after this value matching the job schedule entry name specified will be returned in the list. For more information on using this value to process a partial list, see "Partial List Considerations" on page 2-11.

Error code
I/O; CHAR(*)
The structure in which to return error information. For

the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated Lists

The file member list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section:
  - SCDL0100 format
  - SCDL0200 format

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 63-15.

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)	Job schedule entry name
44	2C	CHAR(16)	Continuation handle

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job schedule entry name used
10	A	CHAR(16)	Continuation handle

SCDL0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(1)	Information status
1	1	CHAR(10)	Job name
11	B	CHAR(10)	Entry number
21	15	CHAR(10)	Scheduled date
31	1F	CHAR(70)	Scheduled days
101	65	CHAR(6)	Scheduled time
107	6B	CHAR(10)	Frequency
117	75	ARRAY(5) of CHAR(10)	Relative day of the month
167	A7	CHAR(10)	Recovery action
177	B1	CHAR(10)	Next submission date
187	BB	CHAR(10)	Status
197	C5	CHAR(10)	Job queue name
207	CF	CHAR(10)	Job queue library name
217	D9	CHAR(10)	User profile of entry adder
227	E3	CHAR(10)	Last submission date
237	ED	CHAR(6)	Last submission time
243	F3	CHAR(50)	Text
293	125	CHAR(23)	Reserved

SCDL0200 Format

Offset		Type	Field
Dec	Hex		
0	0		Returns everything from format SCDL0100
316	13C	CHAR(10)	Job queue status
326	146	ARRAY(20) of CHAR(10)	Dates omitted
526	20E	CHAR(10)	Job description name
536	218	CHAR(10)	Job description library name
546	222	CHAR(10)	User profile for submitted job
556	22C	CHAR(10)	Message queue name
566	236	CHAR(10)	Message queue library name
576	240	CHAR(10)	Save entry
586	24A	CHAR(10)	Last submission job name
596	254	CHAR(10)	Last submission user name
606	25E	CHAR(6)	Last submission job number
612	26E	CHAR(10)	Last attempted submission date
622	26E	CHAR(6)	Last attempted submission time
628	274	CHAR(10)	Status of last attempted submission
638	27E	CHAR(2)	Reserved
640	280	BINARY(4)	Length of command string
644	284	CHAR(512)	Command

Field Descriptions

Command. A command that runs in the submitted batch job if the routing program used when this batch job is started is the IBM-supplied default routing program QCMD. Because this command is used for the request data, this parameter takes the place of any value specified for the RQSDTA parameter in the job description.

Continuation handle. The value used to process a partial list. This value is put in the header section when a partial list is returned. For more information on processing a partial list, see "Partial List Considerations" on page 2-11.

Dates omitted. Specifies up to 20 dates when the job should not be submitted. The dates will be in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century flag indicates the twentieth century, and a 1 indicates the twenty-first century. If no omit dates are specified, this field contains hexadecimal zeros.

Entry number. The entry number assigned to the job schedule entry. The entry number can range from 000001 through 999999.

Format name. The content and format of the information returned for each job schedule entry.

Frequency. How often the job schedule entry is to be submitted to run. Valid values are:

**ONCE* The job schedule entry does not repeat.
**WEEKLY* The job schedule entry is submitted on the same day of each week at the scheduled time.
**MONTHLY* The job schedule entry is submitted on the same day of each month at the scheduled time.

Information status. Whether or not the entry information could be successfully retrieved.

blank No errors occurred. All information is returned for this entry.
A Insufficient authority to entry. Only the SCDL0100 information is returned for this entry. The rest of the entry is blank.
L The entry is locked. Only the SCDL0100 information is returned for this entry. The rest of the entry is blank.

Job description name. The name of the job description used for the job schedule entry.

**USRPRF*
The job description in the user profile under which the job runs is used as the job description of the job schedule entry.
job description name
The name of the job description used for the job schedule entry.

Job description library name. The name of the library in which the job description is located.

Job name. The job name associated with the job schedule entry.

Job queue library name. The name of the library in which the specified job queue resides.

Job queue name. The name of the job queue where the job should be placed when it is submitted. Valid values are:

**JOBD* The job is placed in the job queue named in the specified job description.
job queue name The name of the job queue where the job is placed when it is submitted.

Job queue status. The current status of the job queue. Valid values are:

blank The job queue name specified *JOBD, the job queue could not be found, or the job queue is damaged.
HLD The job queue is held. The job queue is not allocated to a subsystem.
RLS The job queue is released. The job queue is not allocated to a subsystem.
HLD/SBS The job queue is held. The job queue is allocated to a subsystem.
RLS/SBS The job queue is released. The job queue is allocated to a subsystem.
LOCKED The job queue is locked, and the status could not be obtained.

Job schedule entry name used. The job name used to identify the job schedule entry.

Last attempted submission date. The date on which a job could have been last submitted. The value is returned in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century indicates the twentieth century, and a 1 indicates the twenty-first century. If no submission has been attempted, this field contains hexadecimal zeros.

Last attempted submission time. The time at which a job could have been last submitted from this entry. The value is returned in the format HHMMSS, where HH is hours, MM is minutes, and SS is seconds. If no submission has been attempted, this field contains hexadecimal zeros.

Last submission date. The date on which a job was last submitted from this entry. The value is returned in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century indicates the twentieth century, and a 1 indicates the twenty-first century. If there has been no previous submission, this field contains hexadecimal zeros.

Last submission job name. The job name of the last job that was submitted from this entry. If there has been no previous submission, this field contains blanks.

Last submission job number. The job number of the last job that was submitted from this entry. If there has been no previous submission, this field contains blanks.

Last submission time. The time at which a job was last submitted from this entry. The value is returned in the format HHMMSS, where HH is hours, MM is minutes, and SS is seconds. If there has been no previous submission, this field contains hexadecimal zeros.

Last submission user name. The user name of the last job that was submitted from this entry. If there has been no previous submission, this field contains blanks.

Length of command string. The length of the command string specified in the command field.

Message queue name. The name of the message queue, if any, to which a completion message is sent when the job is submitted. A completion message is sent when the submitted job has completed running, and error messages are sent if the Submit Job (SBMJOB) command fails for some reason. Valid values are:

**USRPRF* A completion message is sent to the message queue specified in the user profile associated with the job schedule entry.
**NONE* Messages are not sent to a user-specified message queue. In this case, completion messages are not sent, but error messages are sent to the QSYSOPR message queue.
message queue name The name of the message queue where the messages are sent.

Message queue library name. The library in which the message queue is located.

Next submission date. The next date that a job from this entry is scheduled to be submitted. The next submission date is returned in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century indicates the twentieth century, and a 1 indicates the twenty-first century. If this entry has a status of SAV, this field contains hexadecimal zeros.

Recovery action. The action that will happen if the job cannot be submitted at the designated time because the system is powered down or the system is in the restricted state. The action specified by this parameter then occurs when the system is IPLed or when the system comes out of the restricted state. This parameter does not pertain to the situation where a job was held (by the user) when the designated time elapsed and then released (by the user) at a later time. Also, the recovery action does not pertain to timer events that elapse as a result of changes to the QTIME system value. Valid values are:

**SBMRLS* Submit the job in the released state (RLS).
**SBMHLD* Submit the job in the held state (HLD).
**NOSBM* No job is submitted.

Relative day of the month. The relative day of the month the job should be submitted to run. A total of five values can be returned. If no relative day of the month was specified, this area contains blanks. Valid values are:

1–5 The job should be submitted on the specified day of the week every first, second, third, fourth, or fifth week of the month.
**LAST* The job should be submitted on the last specified day of the week each month.

Reserved. An ignored field.

Save entry. Whether or not an entry that has FRQ(*ONCE) specified should be kept in the job schedule after the job has been submitted.

**NO* This entry will not be kept after the job is submitted.
**YES* This entry will be kept after the job is submitted.

Scheduled date. The date that the job will be submitted. Valid values are:

**CURRENT*
The current date will be used.
**MONTHSTR*
The first day of the month will be used.
**MONTHEND*
The last day of the month will be used.
**NONE* A scheduled date was not specified.
date An actual date in the format CYYMMDD, where C is the century, YY is the year, MM is the month, and DD is the day. A 0 for the century indicates the twentieth century, and a 1 indicates the twenty-first century.

Scheduled days. The day of the week that the job will be submitted. A total of seven values can be returned. Valid values are:

**ALL* The job will be submitted every day. This cannot be specified with any other values.
**NONE* A scheduled day is not specified. This cannot be specified with any other values.
**MON* The job will be submitted on Monday.
**TUE* The job will be submitted on Tuesday.
**WED* The job will be submitted on Wednesday.
**THU* The job will be submitted on Thursday.
**FRI* The job will be submitted on Friday.
**SAT* The job will be submitted on Saturday.
**SUN* The job will be submitted on Sunday.

Scheduled time. The time (on the scheduled date) when the job will be submitted to run. This value is returned in the format HHMMSS, where HH is the hours, MM is the minutes, and SS is the seconds.

Status. The status of the job schedule entry. Valid values are:

SCD The entry is scheduled.
HLD The entry is held.
SAV The entry is saved.

Status of last attempted submission. The action that occurred the last time the system could have submitted a job from this entry. Valid values are:

0 Job not previously submitted.
1 Job successfully submitted.
2 Last job submission failed. Check the message queue for details.
3 Job not submitted due to held status.
4 Job submitted after scheduled time as specified by recovery action.
5 Job not submitted as specified by recovery action.

Text. Text that briefly describes the job schedule entry. If no text was specified, this field contains blanks.

User profile for submitted job. The user profile under which the job will be submitted. Valid values are:

**JOBD* The user profile named in the specified job description is used for the job.
user name
The name of the user profile that is used for the job.

User profile of entry adder. The user profile that created this entry.

User space library name. The library name or special value specified in the call to this API.

User space name. The name of the user space that receives the list.

Error Messages

| CPF1629 E Not authorized to job schedule &1.
| CPF1632 E Job schedule entry &3 number &4 damaged.
| CPF1637 E Job schedule &1 in library &2 in use.
| CPF1640 E Job schedule &1 in library &2 does not exist.
| CPF1641 E Job schedule &1 in library &2 damaged.
| CPF1643 E Job schedule entry name not valid.
| CPF3CF1 E Error code parameter not valid.
| CPF3C21 E Format name &1 is not valid.
| CPF811A E User space &4 in &9 damaged.
| CPF812C E Job schedule &4 in &9 damaged.
| CPF8122 E &8 damage on library &4.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2 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.

List Subsystem Job Queues (QWDLSJBQ) API

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	List format	Input	Char(8)
3	Subsystem and library name	Input	Char(20)
4	Error code	I/O	Char(*)

The List Subsystem Job Queues (QWDLSJBQ) API lists the job queues for a subsystem. It also gives the job queue allocation status, indicating whether the specified subsystem is active and has allocated this job queue or not. QWDLSJBQ replaces any job queue list that already exists in the user space.

Authorities and Locks

User Space Authority *CHANGE
User Space Library Authority
*USE
User Space Lock *EXCLRD
Subsystem Description Authority
*USE
Subsystem Description Library Authority
*READ

Required Parameter Group

Qualified user space name
INPUT; CHAR(20)
The user space that receives the list, and the library in which it is located. The first 10 characters contain the user space name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

List format
INPUT; CHAR(8)
The format to use for the list of job queues. You can use this format name:

SJQL0100 Basic job queue list. For details, see "Format of the Generated List" on page 63-18.

Subsystem and library name
INPUT; CHAR(20)
The subsystem about which to retrieve information, and the library in which the subsystem description is located. The first 10 characters contain the subsystem name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of the Generated List

The list of job queues that the QWDLSJBQ API returns into the user space consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section

The user area and generic header are described in "User Space Format for List APIs" on page 2-7. The remaining items are described in the following sections. For detailed descriptions of the fields in the tables, see "Field Descriptions."

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name
10	A	CHAR(10)	User space library name specified
20	14	CHAR(10)	Subsystem name
30	1E	CHAR(10)	Subsystem library name specified

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Subsystem name
10	A	CHAR(10)	Subsystem library name used

SJQL0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Job queue name
10	A	CHAR(10)	Job queue library name
20	14	BINARY(4)	Sequence number
24	18	CHAR(10)	Allocation indicator
34	22	CHAR(2)	Reserved
36	24	BINARY(4)	Maximum active
40	28	BINARY(4)	Maximum by priority 1
44	2C	BINARY(4)	Maximum by priority 2
48	30	BINARY(4)	Maximum by priority 3
52	34	BINARY(4)	Maximum by priority 4
56	38	BINARY(4)	Maximum by priority 5
60	3C	BINARY(4)	Maximum by priority 6
64	40	BINARY(4)	Maximum by priority 7
68	44	BINARY(4)	Maximum by priority 8
72	48	BINARY(4)	Maximum by priority 9

Field Descriptions

Allocation indicator. A value indicating whether or not the job queue is allocated to the specified subsystem. Valid values are:

**NO* The subsystem has not allocated this job queue. Either this subsystem is inactive, or another subsystem has allocated the job queue.
**YES* The subsystem is active and has allocated this job queue.

Job queue library name. The name of the library in which the specified job queue resides.

Job queue name. The name of a job queue specified in a subsystem description job queue entry.

Maximum active. The maximum number of jobs that can be active at the same time through this job queue entry.

Maximum by priority 1 through 9. The maximum number of jobs that can be active at the same time for each priority level (1 through 9). A -1 in this field indicates that the value is *NOMAX.

Reserved. An ignored field.

Sequence number. The job queue entry sequence number. The subsystem uses this number to determine the order in which job queues are processed. Jobs from the queue with the lowest sequence number are processed first.

Subsystem library name specified. The name or special value specified in the call to this API for the library in which the subsystem description resides.

Subsystem library name used. The name of the library in which the subsystem description resides.

Subsystem name. The name of the active subsystem about which information is being returned.

User space library name specified. The library name or special value specified in the call to this API.

User space name. The name of the user space that receives the list.

Error Messages

CPF1605 E Cannot allocate subsystem description &1.
CPF1606 E Error during allocation of subsystem &1.
CPF1607 E Previous request pending for subsystem &1.
CPF1608 E Subsystem description &1 not found.
CPF1619 E Subsystem description &1 in library &2 damaged.
CPF1835 E Not authorized to subsystem description.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF811A E User space &4 in &9 damaged.
CPF8122 E &8 damage on library &4.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9838 E User profile storage limit exceeded.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Data Area (QWCRDTAA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Qualified data area name	Input	Char(20)
4	Starting position	Input	Binary(4)
5	Length of data	Input	Binary(4)
6	Error code	I/O	Char(*)

The Retrieve Data Area (QWCRDTAA) API allows you to retrieve the contents of a data area.

Authorities and Locks

Library Authority *USE
Data Area Authority *USE
Data Area Lock *SHRRD

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that will receive the contents of the data area being retrieved. For the format, see "Format of Data Returned."

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable described in "Format of Data Returned." If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Qualified data area name
INPUT; CHAR(20)
The first 10 characters contain the data area name, and the second 10 characters contain the name of the library where the data area is located.

When one of the special values is specified, the library name must be blank. The special values for the data area are:

***LDA** Local data area
***GDA** Group data area
***PDA** Program initialization parameter data area

The special values supported for the library name are:

***LIBL** The library list.
***CURLIB** The job's current library.

Starting position
INPUT; BINARY(4)
The first byte of the data area to be retrieved. A value of 1 will identify the first character in the data area. The maximum value allowed for the starting position is 2000. A value of -1 will return all the characters in the data area.

Length of data
INPUT; BINARY(4)
The length of the data to be retrieved. This length must not be larger than the size of the variable that is to receive the data. It must also be greater than 0.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of Data Returned

The receiver variable holds the information returned for the data area. The following table shows the format of the receiver variable.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	CHAR(10)	Type of value returned
18	12	CHAR(10)	Library name
28	1C	BINARY(4)	Length of value returned
32	20	BINARY(4)	Number of decimal positions

Offset			
Dec	Hex	Type	Field
36	24	CHAR(*)	Value

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 receiver variable was not sufficiently large to hold all of the data available, this value will be less than the bytes available.

Length of value returned. The length of the value that was returned.

Library name. The name of the library where the data area was found. This field will be blank if one of the special values was specified for the first ten character of the queue data area name.

Number of decimal positions. The number of decimal positions.

Type of value returned. The following values may be returned.

**CHAR* A character data area.
**DEC* A decimal data area. The value returned will be a packed decimal value.
**LGL* A logical data area.

Value. The contents of the data area.

Error Messages

CPF1015 E Data area not found.
CPF1016 E User not authorized to data area.
CPF1021 E Library not found.
CPF1022 E User not authorized to read from library.
CPF1046 E *GDA only valid for group jobs.
CPF1063 E Lock could not be obtained on data area.
CPF1067 E Lock could not be obtained on library.
CPF1072 E *PDA only valid for prestart job.
CPF1088 E Invalid substring starting position.
CPF1089 E Invalid substring specification.
CPF1863 E Length of data not valid.
CPF24B4 E Severe error addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error with receiver variable.
CPF3C24 E Receiver variable length not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Job Description Information (QWDRJOBD) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified job description name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Job Description Information (QWDRJOBD) API retrieves information from a job description object and places it into a single variable in the calling program. The amount of information returned depends on the size of the variable. The information returned is the same information returned by the Display Job Description (DSPJOBD) command.

Authorities and Locks

Job Description Object Authority *USE
Library Authority *USE

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. You can specify the size of this area to be smaller than the format requested if you specify the length of receiver variable parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If this value is larger than the actual size of the receiver variable, the result may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the job description information to be returned. You can use this format:

JOBD0100 Basic job description information. For details, see "JOBD0100 Format" on page 63-21.

Qualified job description name
INPUT; CHAR(20)
The name of the job description whose contents are to be retrieved. The first 10 characters contain the name of the job description, and the second 10 characters contain the name of the library where the job description is located. You can use these special values for the library name:

***CURLIB** The job's current library

***LIBL** The library list

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

JOBD0100 Format

The following table describes the information that is returned in the receiver variable for the JOBD0100 format. For detailed descriptions of the fields, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Job description name
18	12	CHAR(10)	Job description library name
28	1C	CHAR(10)	User name
38	26	CHAR(8)	Job date
46	2E	CHAR(8)	Job switches
54	36	CHAR(10)	Job queue name
64	40	CHAR(10)	Job queue library name
74	4A	CHAR(2)	Job queue priority
76	4C	CHAR(10)	Hold on job queue
86	56	CHAR(10)	Output queue name
96	60	CHAR(10)	Output queue library name
106	6A	CHAR(2)	Output queue priority
108	6C	CHAR(10)	Printer device name
118	76	CHAR(30)	Print text
148	94	BINARY(4)	Syntax check severity
152	98	BINARY(4)	End severity
156	9C	BINARY(4)	Message logging severity
160	A0	CHAR(1)	Message logging level
161	A1	CHAR(10)	Message logging text
171	AB	CHAR(10)	Logging of CL programs
181	B5	CHAR(10)	Inquiry message reply
191	BF	CHAR(13)	Device recovery action
204	CC	CHAR(10)	Time-slice end pool
214	D6	CHAR(15)	Accounting code
229	E5	CHAR(80)	Routing data
309	135	CHAR(50)	Text description
359	167	CHAR(1)	Reserved
360	168	BINARY(4)	Offset to initial library list
364	16C	BINARY(4)	Number of libraries in initial library list
368	170	BINARY(4)	Offset to request data
372	174	BINARY(4)	Length of request data
376	174	CHAR(*)	Reserved
*	*	ARRAY (*) of CHAR(11)	Initial library list
*	*	CHAR(*)	Request data

Field Descriptions

Accounting code. An identifier assigned to jobs that use this job description. This code is used to collect system resource use information. If the special value *USRPRF is specified, the accounting code used for jobs using this job description is obtained from the job's user profile.

Bytes available. The length of all data available to return. All available data is returned if enough space is provided.

Bytes returned. The length of all data actually returned. If the data is truncated because the receiver variable was not sufficiently large to hold all of the data available, this value will be less than the bytes available.

Device recovery action. The action to take when an I/O error occurs for the interactive job's requesting program device. The possible values are:

**SYSVAL*
The value in the system value QDEVRCYACN at the time the job is started is used as the device recovery action for this job description.

**MSG* Signals the I/O error message to the application and lets the application program perform error recovery.

**DSCMSG*
Disconnects the job when an I/O error occurs. When the job reconnects, the system sends a message to the application program, indicating the job has reconnected and that the work station device has recovered.

**DSCENDRQS*
Disconnects the job when an I/O error occurs. When the job reconnects, the system sends the End Request (ENDRQS) command to return control to the previous request level.

**ENDJOB*
Ends the job when an I/O error occurs. A message is sent to the job's log and to the history log (QHST) indicating the job ended because of a device error.

**ENDJOBNOLIST*
Ends the job when an I/O error occurs. There is no job log produced for the job. The system sends a message to the history log (QHST) indicating the job ended because of a device error.

End severity. The message severity level of escape messages that can cause a batch job to end. The batch job ends when a request in the batch input stream sends an escape message, whose severity is equal to or greater than this value, to the request processing program. The possible values are from 0 through 99.

Hold on job queue. Whether jobs using this job description are put on the job queue in the hold condition. The possible values are *YES and *NO.

Initial library list. The initial library list that is used for jobs that use this job description. Only the libraries in the user portion of the library list are included.

Note: The data is an array of 11-byte entries, each entry consisting of a 10-byte library name left-justified with a blank pad at the end. The number of libraries in the initial library list tells how many entries are contained in the array.

Inquiry message reply. How inquiry messages are answered for jobs that use this job description.

**RQD* The job requires an answer for any inquiry messages that occur while the job is running.

**DFT* The system uses the default message reply to answer any inquiry messages issued while the job is running. The default reply is either defined in the message description or is the default system reply.

**SYSRPYL*
The system reply list is checked to see if there is an entry for an inquiry message issued while the job is running. If a match occurs, the system uses the reply value for that entry. If no entry exists for that message, the system uses an inquiry message.

Job date. The date that will be assigned to jobs using this job description when they are started. The possible values are:

**SYSVAL*
The value in the QDATE system value is used at the time the job is started.

job-date The date to be used at the time the job is started. This date is in the format specified for the DATFMT job attribute.

Job description library name. The name of the library in which the job description resides.

Job description name. The name of the job description about which information is being returned.

Job queue library name. The library of the job queue into which batch jobs using this job description are placed.

Job queue name. The name of the job queue into which batch jobs using this job description are placed.

Job queue priority. The scheduling priority of each job that uses this job description. The highest priority is 1 and the lowest priority is 9.

Job switches. The initial settings for a group of eight job switches used by jobs that use this job description. These switches can be set or tested in a program and used to control a program's flow. The possible values are '0' (off) and '1' (on).

Length of request data. The length of all available request data, in bytes. If the receiver variable was not sufficiently large to hold all of the request data available, the amount of request data actually returned may be less than this value.

Logging of CL programs. Whether or not messages are logged for CL programs that are run. The possible values are *YES and *NO.

Message logging level. The type of information logged. Possible types are:

0 No messages are logged.

1 All messages sent to the job's external message queue with a severity greater than or equal to the message logging severity are logged.

2 The following information is logged:
- Level 1 information.
- Requests or commands from CL programs for which the system issues messages with a severity code greater than or equal to the logging severity.
- All messages associated with those requests or commands that have a severity code greater than or equal to the logging severity.

3 The following information is logged:
- Level 1 information.
- All requests or commands from CL programs.
- All messages associated with those requests or commands that have a severity greater than or equal to the logging severity.

4 The following information is logged:
- All requests or commands from CL programs.
- All messages with a severity code greater than or equal to the logging severity.

Message logging severity. The minimum severity level that causes error messages to be logged in the job log. The possible values are from 0 through 99.

Message logging text. The level of message text that is written in the job log or displayed to the user when an error message is created according to the logging level and logging severity. The possible values are:

**MSG* Only the message is written to the job log.

**SECLVL*
Both the message and the message help for the error message are written to the job log.

**NOLIST*
If the job ends normally, there is no job log. If the job ends abnormally (the job end code is 20 or

higher), there is a job log. The messages appearing in the job's log contain both the message and the message help.

Number of libraries in initial library list. The number of libraries in the user portion of the initial library list. Up to 25 libraries can specified.

Offset to initial library list. The offset from the beginning of the structure to the start of the initial library list.

Offset to request data. The offset from the beginning of the structure to the start of the request data.

Output queue library name. The name of the library in which the output queue resides.

Output queue name. The name of the default output queue that is used for spooled output produced by jobs that use this job description.

**USRPRF* The output queue name for jobs using this job description is obtained from the user profile of the job at the time the job is started.

**DEV* The output queue with the same name as the printer device for this job description is used.

**WRKSTN*
The output queue name is obtained from the device description from which this job is started.

output-queue-name
The name of the output queue for this job description.

Output queue priority. The output priority for spooled files that are produced by jobs using this job description. The highest priority is 1, and the lowest priority is 9.

Print text. The line of text (if any) that is printed at the bottom of each page of printed output for jobs using this job description. If the special value *SYSVAL is specified, the value in the system value QPRTTXT is used for jobs using this job description.

Printer device name. The name of the printer device or the source for the name of the printer device that is used for all spooled files created by jobs that use this job description.

**USRPRF*
The printer device name is obtained from the user profile of the job at the time the job is started.

**SYSVAL*
The value in the system value QPRTDEV at the time the job is started is used as the printer device name.

**WRKSTN*
The printer device name is obtained from the work station where the job was started.

printer-device-name
The name of the printer device that is used with this job description.

Request data. The request data that is placed as the last entry in the job's message queue for jobs that use this job description. The possible values are:

**NONE* No request data is placed in the job's message queue.

**RTGDTA*
The data specified in the routing data parameter is placed as the last entry in the job's message queue.

request-data
The request data to use for jobs that use this job description.

Reserved. An ignored field.

Routing data. The routing data that is used with this job description to start jobs. The possible values are:

QCMDI The default routing data QCMDI is used by the IBM-supplied interactive subsystem to route the job to the IBM-supplied control language processor QCMD in the QSYS library.

**RQSDTA*
Up to the first 80 characters of the request data specified in the request data field are used as the routing data for the job.

routing-data
The routing data to use for jobs that use this job description.

Syntax check severity. Whether requests placed on the job's message queue are checked for syntax as CL commands, and the message severity that causes a syntax error to end processing of a job. The possible values are:

-1 The request data is not checked for syntax as CL commands. This is equivalent to *NOCHK.

0–99 Specifies the lowest message severity that causes a running job to end. The request data is checked for syntax as CL commands, and, if a syntax error occurs that is greater than or equal to the error message severity specified here, the running of the job that contains the erroneous command is suppressed.

Text description. The user text, if any, used to briefly describe the job description.

Time-slice end pool. Whether interactive jobs using this job description should be moved to another main storage pool when they reach time-slice end. The possible values are:

**SYSVAL* The system value is used.

**NONE* The job is not moved when it reaches time-slice end.

**BASE* The job is moved to the base pool when it reaches time-slice end.

User name. The name of the user profile associated with this job description. If *RQD is specified, a user name is required to use the job description.

Error Messages

| CPF1618 E Job description &1 in library &2 damaged.
| CPF24B4 E Severe error occurred while addressing parameter list.
| CPF3CF1 E Error code parameter not valid.
| CPF3CF2 E Error(s) occurred during running of &1 API.
| CPF3C21 E Format name &1 not valid.
| CPF3C24 E Length of receiver variable not valid.
| CPF9801 E Object &2 in library &3 not found.
| CPF9802 E Not authorized to object &2.
| CPF9803 E Cannot allocate object &2 in library &3.
| CPF9804 E Object &2 in library &3 damaged.
| CPF9807 E One or more libraries in library list deleted.
| CPF9808 E Cannot allocate one or more libraries on library list.
| CPF9810 E Library &1 not found.
| CPF9820 E Not authorized to use library &1.
| CPF9830 E Cannot assign library &1.
| CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Job Information (QUSRJOBI) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Qualified job name	Input	Char(26)
5	Internal job identifier	Input	Char(16)

Optional Parameter:

6	Error code	I/O	Char(*)

The Retrieve Job Information (QUSRJOBI) API retrieves specific information about a job.

Authorities and Locks

Job Authority *JOBCTL, if the job for which information is retrieved has a different user profile from that of the job that calls the QUSRJOBI API.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. You can specify the size of this area to be smaller than the format requested as long as you specify the length parameter correctly. As a result, the API returns only the data that the area can hold.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the job information to be returned. The format names supported are:

JOBI0100 Basic performance information
JOBI0150 Additional performance information
JOBI0200 WRKACTJOB information
JOBI0300 Job queue and output queue information
JOBI0400 Job attribute information
JOBI0500 Message logging information
JOBI0600 Active job information
JOBI0700 Library list information

Refer to "Selecting a Job Information Format" on page 63-25 for details of each of the formats.

Qualified job name
INPUT; CHAR(26)
The name of the job for which information is to be returned. The qualified job name has three parts:

Job name CHAR(10). A specific job name or one of the following special values:
- * The job that this program is running in. The rest of the qualified job name parameter must be blank.
- ***INT** The internal job identifier locates the job. The user name and job number must be blank.

User name CHAR(10). A specific user profile name, or blanks when the job name is a special value or *INT.

Job number CHAR(6). A specific job number, or blanks when the job name specified is a special value or *INT.

Internal job identifier
INPUT; CHAR(16)
The internal identifier for the job. The List Job API, QUSLJOB, creates this identifier. If you do not specify *INT for the job name parameter, this parameter must contain blanks. With this parameter, the system can locate the job more quickly than with a job name.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Selecting a Job Information Format

The following section presents some of the performance characteristics of the different formats (primarily JOBI0100, JOBI0150, and JOBI0200). When formats return some of the same information, the performance effects are discussed. When a format contains information not available in other formats, performance is not discussed.

JOBI0100 This format returns basic performance information about a job. It is faster than the JOBI0150 format and the JOBI0200 format (which also contain performance information). The reason that this format is faster is that it does not touch as many objects, causing less paging when retrieving information about the job.

JOBI0150 This format returns additional performance information, and is slower than the JOBI0100 format. It is similar to the JOBI0200 format, but is faster than that format because there is less paging involved in retrieving the information.

JOBI0200 This format returns information equivalent to that found on the Work with Active Jobs (WRKACTJOB) command.

JOBI0300 This format returns job queue and output queue information for a job, as well as information about the submitter's job if the job is a submitted batch job.

JOBI0400 This format primarily returns job attribute types of information, but has other types of information as well.

JOBI0500 This format returns message logging information.

JOBI0600 This format returns information about active jobs only. It is intended to supplement the JOBI0400 format. It retrieves information from several additional objects associated with the job, and therefore, it causes additional paging.

JOBI0700 This format returns library list information for an active job.

Each format returns information that is only valid for the status of certain jobs. For example, the JOBI0200 format only returns information for active jobs. Because the job status can change between the time the list is generated and the time the Retrieve Job Information API is called, you must design your application to handle this.

When requesting information about a job that has an unknown or incorrect job status for the format requested, the API returns the current status of the job and sets the remainder of the fields for that format to zeros and blanks. When requesting information about a job that is not valid, the API returns the job's status as blanks and sets the remainder of the fields for that format to zeros and blanks. Therefore, you should check the returned status of the job before processing the data. Each format description specifies each status for which the API returns complete information.

JOBI0100 Format

The JOBI0100 format information is valid for active jobs and jobs on queues. For jobs on queues, this format returns zeros or blanks for the attributes. If the Change Job (CHGJOB) command was run against a job on a *JOBQ, the attributes returned are the attributes specified on the CHGJOB command. If the job status changes to *OUTQ, the status field returned is *OUTQ and the API returns no information other than the number of bytes returned, the number of bytes available, the qualified job name, the job type, the job subtype, and the internal job identifier.

The JOBI0100 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(2)	Reserved
64	40	BINARY(4)	Run priority
68	44	BINARY(4)	Time slice
72	48	BINARY(4)	Default wait
76	4C	CHAR(10)	Purge

For more details about the fields listed in the previous table, see "Field Descriptions" on page 63-28.

JOBI0150 Format

The JOBI0150 format is valid for active jobs only. If the job status changes to *OUTQ or *JOBQ, the status field is set appropriately, and no information other than the number of bytes returned, the number of bytes available, the qualified job name, the job type, the job subtype, and the internal job identifier is returned.

The JOBI0150 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	&rblk.	Returns everything from format JOBI0100
86	56	CHAR(10)	Time-slice end pool
96	60	BINARY(4)	Processing unit used

Offset		Type	Field
Dec	Hex		
100	64	BINARY(4)	System pool identifier
104	68	BINARY(4)	Maximum processing unit time
108	6C	BINARY(4)	Temporary storage used
112	70	BINARY(4)	Maximum temporary storage

For more details about the fields listed in the previous table, see "Field Descriptions" on page 63-28.

JOBI0200 Format

The JOBI0200 format is only valid for active jobs and is similar to the information supported by the Work with Active Jobs (WRKACTJOB) command. If the job status has changed to *OUTQ or *JOBQ, the status field is set appropriately, and no information other than the number of bytes returned, the number of bytes available, the qualified job name, the job type, the job subtype, and the internal job identifier is returned.

The JOBI0200 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(10)	Subsystem description name
72	48	BINARY(4)	Run priority
76	4C	BINARY(4)	System pool identifier
80	50	BINARY(4)	Processing unit used
84	54	BINARY(4)	Number of auxiliary I/O requests
88	58	BINARY(4)	Number of interactive transactions
92	5C	BINARY(4)	Response time total
96	60	CHAR(1)	Function type
97	61	CHAR(10)	Function name
107	6B	CHAR(4)	Active job status

For more details about the fields listed in the previous table, see "Field Descriptions" on page 63-28.

JOBI0300 Format

This format returns job queue and output queue information for a job, as well as information about the submitter's job. This information is valid for any job status. The JOBI0300 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(10)	Job queue name
72	48	CHAR(10)	Job queue library name
82	52	CHAR(2)	Job queue priority
84	54	CHAR(10)	Output queue name
94	5E	CHAR(10)	Output queue library name
104	68	CHAR(2)	Output queue priority
106	6A	CHAR(10)	Printer device name
116	74	CHAR(10)	Submitter's job name
126	7E	CHAR(10)	Submitter's user name
136	88	CHAR(6)	Submitter's job number
142	8E	CHAR(10)	Submitter's message queue name
152	98	CHAR(10)	Submitter's message queue library name
162	A2	CHAR(10)	Status of job on the job queue
172	AC	CHAR(8)	Date and time job was put on this job queue
180	B4	CHAR(7)	Job date

For more details about the fields listed in the previous table, see "Field Descriptions" on page 63-28.

JOBI0400 Format

This format primarily returns job attribute types of information, but has other types of information as well. This format is valid for any job status. The JOBI0400 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(13)	Date and time job entered the system
75	4B	CHAR(13)	Date and time job became active
88	58	CHAR(15)	Job accounting code
103	67	CHAR(10)	Job description name
113	71	CHAR(10)	Job description library name
123	7B	CHAR(24)	Unit of work ID
147	93	CHAR(8)	Mode name
155	9B	CHAR(10)	Inquiry message reply
165	A5	CHAR(10)	Logging of CL programs
175	AF	CHAR(10)	Break message handling
185	B9	CHAR(10)	Status message handling
195	C3	CHAR(13)	Device recovery action
208	D0	CHAR(10)	DDM conversation handling
218	DA	CHAR(1)	Date separator
219	DB	CHAR(4)	Date format
223	DF	CHAR(30)	Print text
253	FD	CHAR(10)	Submitter's job name
263	107	CHAR(10)	Submitter's user name
273	111	CHAR(6)	Submitter's job number
279	117	CHAR(10)	Submitter's message queue name
289	121	CHAR(10)	Submitter's message queue library name
299	12B	CHAR(1)	Time separator
300	12C	Binary(4)	Coded character set ID
304	130	CHAR(8)	Date and time the job is scheduled to run
312	138	CHAR(10)	Print key format
322	142	CHAR(10)	Sort sequence
332	14C	CHAR(10)	Sort sequence library
342	156	CHAR(3)	Language ID
345	159	CHAR(2)	Country ID
347	15B	CHAR(1)	Completion status
348	15C	CHAR(1)	Signed-on job

For more details about the fields listed in the previous table, see "Field Descriptions" on page 63-28.

JOBI0500 Format

This format returns message logging information. This format is valid for any job status. The JOBI0500 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(2)	Reserved
64	40	BINARY(4)	End severity
68	44	BINARY(4)	Logging severity
72	48	CHAR(1)	Logging level
73	49	CHAR(10)	Logging text

For more details about the fields listed in the previous table, see "Field Descriptions" on page 63-28.

JOBI0600 Format

The JOBI0600 format returns information about active jobs. If the job status changes to *JOBQ or *OUTQ, the status field is set appropriately, and no information other than the number of bytes returned, the number of bytes available, the qualified job name, the job type, the job subtype, and the internal job identifier is returned.

The JOBI0600 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type

Offset		Type	Field
Dec	Hex		
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(8)	Job switches
70	46	CHAR(1)	End status
71	47	CHAR(10)	Subsystem description name
81	51	CHAR(10)	Subsystem description library name
91	5B	CHAR(10)	Current user profile
101	65	CHAR(1)	DBCS-capable
102	66	CHAR(1)	Exit key
103	67	CHAR(1)	Cancel key
104	68	BINARY(4)	Product return code
108	6C	BINARY(4)	User return code
112	70	BINARY(4)	Program return code
116	74	CHAR(10)	Special environment

For more details about the fields listed in the previous table, see "Field Descriptions."

JOBI0700 Format

The JOBI0700 format returns library list information for active jobs only. The format returns the actual length instead of the total length because all libraries may not exist. The JOBI0700 format returns the following job information.

Offset		Type	Field
Dec	Hex		
0	0	BINARY(4)	Number of bytes returned
4	4	BINARY(4)	Number of bytes available
8	8	CHAR(10)	Job name
18	12	CHAR(10)	User name
28	1C	CHAR(6)	Job number
34	22	CHAR(16)	Internal job identifier
50	32	CHAR(10)	Job status
60	3C	CHAR(1)	Job type
61	3D	CHAR(1)	Job subtype
62	3E	CHAR(2)	Reserved
64	40	BINARY(4)	Number of libraries in SYSLIBL
68	44	BINARY(4)	Number of product libraries
72	48	BINARY(4)	Current library existence
76	4C	BINARY(4)	Number of libraries in USRLIBL
See note	See note	Array(*) of CHAR(11)	System library list (for each library in the list)
See note	See note	Array(*) of CHAR(11)	Product libraries if they exist
See note	See note	Array(*) of CHAR(11)	Current library if one exists
See note	See note	Array(*) of CHAR(11)	User library list (for each library in the list)

Note: The decimal and hexadecimal offsets depend on the number of libraries you have in the various parts of your library lists. The data is left-justified with a blank pad at the end. The array is sequential. It is an array or data structure. See the *CL Programmer's Guide* for the total number of libraries that can be returned to you.

For more details about the fields listed in the previous table, see "Field Descriptions."

Field Descriptions

The following section describes the fields returned in further detail. The fields are listed in alphabetical order.

Active job status. The status of the job. Each job displays only one status. The possible values are:

no status A blank status field represents a job that is in transition or is not active.

BSCA Waiting in a pool activity level for the completion of an I/O operation to a binary synchronous device.

BSCW Waiting for the completion of an I/O operation to a binary synchronous device.

CMNA Waiting in a pool activity level for the completion of an I/O operation to a communications device.

CMNW Waiting for the completion of an I/O operation to a communications device.

CMTW Waiting for the completion of save-while-active checkpoint processing in another job.

CPCW Jobs waiting for the completion of a CPI Communications call.

DEQA Waiting in the pool activity level for completion of a dequeue operation.

DEQW Waiting for completion of a dequeue operation. For example, QSYSARB and subsystem monitors generally wait for work by waiting for a dequeue operation.

DKTA Waiting in a pool activity level for the completion of an I/O operation to a diskette unit.

DKTW Waiting for the completion of an I/O operation to a diskette unit.

DLYW The Delay Job (DLYJOB) command delays the job for a time interval to end, or for a specific delay end time. The function field shows either the number of seconds the job is to delay (999999), or the specific time when the job is to resume running.

DSC Disconnected from a work station display.

DSPA Waiting in a pool activity level for input from a work station display.
DSPW Waiting for input from a work station display.
END The job has been ended with the *IMMED option, or its delay time has ended with the *CNTRLD option.
EOFA Waiting in the activity level to try a read operation again on a database file after the end-of-file has been reached.
EOFW Waiting to try a read operation again on a database file after the end-of-file has been reached.
EOJ Ending for a reason other than running the End Job (ENDJOB) or End Subsystem (ENDSBS) command, such as SIGNOFF, End Group Job (ENDGRPJOB), or an exception that is not handled.
EVTW Waiting for an event. For example, QLUS and SCPF generally wait for work by waiting for an event.
GRP Suspended by a Transfer Group Job (TFRGRPJOB) command.
HLD Held.
ICFA Waiting in a pool activity level for the completion of an I/O operation to an intersystem communications function file.
ICFW Waiting for the completion of an I/O operation to an intersystem communications function file.
INEL Ineligible and not currently in the pool activity level.
LCKW Waiting for a lock.
MLTA Waiting in a pool activity level for the completion of an I/O operation to multiple files.
MLTW Waiting for the completion of an I/O operation to multiple files.
MSGW Waiting for a message from a message queue.
MXDW Waiting for the completion of an I/O operation to a mixed device file.
OSIW Jobs waiting for the completion of an OSI Communications Subsystem/400 OSLISN, OSRACS, OSRACA, OSRCV, or OSRCVA operation.
PRTA Waiting in a pool activity level for output to a printer to complete.
PRTW Waiting for output to a printer to be completed.
PSRW A prestart job waiting for a program start request.
RUN Currently running in the pool activity level.
SRQ The suspended half of a system request job pair.
SVFA Waiting in a pool activity level for completion of a save file operation.
SVFW Waiting for completion of a save file operation.
TAPA The job is waiting in a pool activity level for completion of an I/O operation to a tape device.
TAPW Waiting for completion of an I/O operation to a tape device.
TIMA Waiting in a pool activity level for a time interval to end.
TIMW Waiting for a time interval to end.

Break message handling. How this job handles break messages. The possible values are:

**NORMAL* The message queue status determines break message handling.
**HOLD* The message queue holds break messages until a user or program requests them. The work station user uses the Display Message (DSPMSG) command to display the messages; a program must issue a Receive Message (RCVMSG) command to receive a message and handle it.
**NOTIFY* The system notifies the job's message queue when a message arrives. For interactive jobs, the audible alarm sounds if there is one, and the message-waiting light comes on.

Cancel key. Whether the user pressed the Cancel key.

0 The user did not press the Cancel key.
1 The user did press the Cancel key.

Note: The application or command that was called before this API determines how the key is set.

Coded character set ID. The coded character set identifier used for this job.

Completion status. The completion status of the job.

blank The job has not completed.
0 The job completed normally.
1 The job completed abnormally.

Country ID. The country identifier associated with this job.

Current library existence. The current library existence field:

0 No current library exists.
1 A current library exists.

Current library if one exists. The name of the current library for the job. If no current library exists, the current library existence field is zero and this field has no entry in the list.

Current user profile. The user profile that the job for which information is being retrieved is currently running under. This name may differ from the user portion of the job name.

Date and time job became active. When the job began to run on the system. This is blank if the job did not become active. It is in the format CYYMMDDHHMMSS, where:

C Century. 0 is the twentieth century, and 1 is the twenty-first century.
YY Year
MM Month
DD Day
HH Hour
MM Minute
SS Second

Date and time job entered system. When the job was placed on the system, in the CYYMMDDHHMMSS format described for the date and time job became active field.

Date and time job is scheduled to run. Date and time the job is scheduled to become active. This field is blank if the job is not a scheduled job. The format for this field is the system time-stamp format.

Date and time the job was put on this job queue. This is the date and time this job was put on this job queue. It is in system timestamp format. This field will contain blanks if the job was not on a job queue.

Date format. The format that the date is presented in. The following values are possible:

**YMD*	Year, month, and day format
**MDY*	Month, day, and year format
**DMY*	Day, month, and year format
**JUL*	Julian format (year and day)

Date separator. The value used to separate days, months, and years when presenting a date.

DBCS-capable. Whether the job is DBCS-capable.

0	The job is not DBCS-capable.
1	The job is DBCS-capable.

DDM conversation handling. Whether the distributed data management (DDM) conversations are *KEEP or *DROP when they are not used.

Default wait. The default maximum time (in seconds) that a job waits for a system instruction that performs a wait function, such as a LOCK machine interface (MI) instruction, to complete running. A value of -1 is returned if the value is *NOMAX.

Device recovery action. The action taken for interactive jobs when an I/O error occurs for the job's requesting program device. The possible values are:

**MSG*
: Signals the I/O error message to the application and lets the application program perform error recovery.

**DSCMSG*
: Disconnects the job when an I/O error occurs. When the job reconnects, the system sends an error message to the application program, indicating the job has reconnected and that the work station device has recovered.

**DSCENDRQS*
: Disconnects the job when an I/O error occurs. When the job reconnects, the system sends the End Request (ENDRQS) command to return control to the previous request level.

**ENDJOB*
: Ends the job when an I/O error occurs. A message is sent to the job's log and to the history log (QHST) indicating the job ended because of a device error.

**ENDJOBNOLIST*
: Ends the job when an I/O error occurs. There is no job log produced for the job. The system sends a message to the QHST log indicating the job ended because of a device error.

End severity. The message severity level of escape messages that can cause a batch job to end. The batch job ends when a request in the batch input stream sends an escape message, whose severity is equal to or greater than this value, to the request processing program.

End status. Whether or not the system issued a controlled cancelation. The possible values are:

1	The system, the subsystem in which the job is running, or the job itself is canceled.
0	The system, subsystem, or job is not canceled.
blank	The job is not running.

Exit key. Whether the user pressed the Exit key.

0	The user did not press the Exit key.
1	The user did press the Exit key.

Note: The application or command that was called before this API determines how the key is set.

Function name. Additional information (as described in the function type field) about the function the job is currently performing. This information is updated only when a command is processed.

Function type. Whether the job is performing a high-level function and what the function type is. The possible values are:

blank	The system is not doing a logged function.
C	A command is running interactively, or it is in a batch input stream, or it was requested from a system menu. Commands in CL programs or REXX procedures are not logged.
D	The job is processing a Delay Job (DLYJOB) command. The function name contains the number of seconds the job is delayed (up to 999999 seconds), or the time when the job is to resume processing (HH:MM:SS), depending on how you specified the command.
G	The Transfer Group Job (TFRGRPJOB) command suspended the job. The function name field contains the group job name for that job.
I	The job is rebuilding an index (access path). The function name field contains the name of the logical file whose index is rebuilt.
L	The system logs history information in a database file. The function name field contains the name of the log (QHST is the only log currently supported).
M	The job is a multiple requester terminal (MRT) job if the job type is BATCH and the subtype is MRT, or it is an interactive job attached to an MRT job if the job type is interactive. See job type, subtype, or field for how to determine what type of job this is. For MRT jobs, the function name field contains information in the following format:

- CHAR(2): The number of requesters currently attached to the MRT job.
- CHAR(1): The field is reserved for a / (slash).
- CHAR(2): The maximum number (MRTMAX) of requesters.
- CHAR(1): Reserved.
- CHAR(3): The never-ending program (NEP) indicator. If an MRT is also an NEP, the MRT stays active even if there are no requesters of the MRT. A value of NEP indicates a never-ending program. A value of blanks indicates that it is not a never-ending program.
- CHAR(1): Reserved.

For interactive jobs attached to an MRT, the function name field contains the name of the MRT procedure.

N The job is currently at a system menu. The function name field contains the name of the menu.

O The job is a subsystem monitor that is performing input/output (I/O) operations to a work station. The function name field contains the name of the work station device to which the subsystem is performing an input/output operation.

P The job is running a program. The function name field contains the name of the program.

R The job is running a procedure. The function name field contains the name of the procedure.

* This does a special function. For this value, the function name field contains one of the following values:

- ADLACTJOB: Auxiliary storage is being allocated for the number of active jobs specified in the QADLACTJ system value. This may indicate that the system value for the initial number of active jobs is too low.
- ADLTOTJOB: Auxiliary storage is being allocated for the number of jobs specified in the QADLTOTJ system value.
- CMDENT: The Command Entry display is being used.
- DIRSHD: Directory shadowing.
- DLTSPLF: The system is deleting a spooled file.
- DUMP: A dump is in process.
- JOBLOG: The system is producing a job log.
- PASSTHRU: The job is a pass-through job.
- RCLSPLSTG: Empty spooled database members are being deleted.
- SPLCLNUP: Spool cleanup is in process.

Inquiry message reply. How the job answers inquiry messages:

**RQD* The job requires an answer for any inquiry messages that occur while this job is running.

**DFT* The system uses the default message reply to answer any inquiry messages issued while this job is running. The default reply is either defined in the message description or is the default system reply.

**SYSRPYL* The system reply list is checked to see if there is an entry for an inquiry message issued while this job is running. If a match occurs, the system uses the reply value for that entry. If no entry exists for that message, the system uses an inquiry message.

Internal job identifier. A value input to other APIs to increase the speed of locating the job on the system. Only APIs described in this manual use this identifier. The identifier is not valid following an initial program load (IPL). If you attempt to use it after an IPL, an exception occurs.

Job accounting code. An identifier assigned to the job by the system to collect resource use information for the job when job accounting is active.

Job date. This is the date to be used for the job. It is in the format CYYMMDD where C is the century, YY is the year, MM is the month, and DD is the day. This value is for jobs whose status is *JOBQ or *ACTIVE. For jobs with a status of *OUTQ, the value for this field is blank.

**SYSVAL* This job will use the system date.

Job description library name. The library containing the job description.

Job description name. A set of job-related attributes used for one or more jobs on the system. These attributes determine how the job is run on the system. Multiple jobs can also use the same job description.

Job name. The name of the job as identified to the system. For an interactive job, the system assigns the job the name of the work station where the job started; for a batch job, you specify the name in the command when you submit the job.

Job number. The system-generated job number.

Job queue library name. The name of the library where the job queue is located.

Job queue name. The name of the job queue that the job is currently on, or that the job is on when it is currently active. This value is for jobs whose status is *JOBQ or *ACTIVE. For jobs with a status of *OUTQ, the value for this field is blank.

Job queue priority. The scheduling priority of the job compared to other jobs on the same job queue. The highest priority is 0 and the lowest is 9. This value is for jobs whose

status is *JOBQ or *ACTIVE. For jobs with a status of *OUTQ, the value for this field is blank.

Job status. The status of the jobs. The special values are:

**ACTIVE* Active jobs. This includes group jobs, system request jobs, and disconnected jobs.
**ALL* All jobs, regardless of status.
**JOBQ* Jobs that are currently on job queues.
**OUTQ* Jobs that have completed running but still have output on an output queue.

Job subtype. Additional information about the job type (if any exists). The possible values are:

blank The job has no special subtype or is not a valid job.
D The job is an immediate job.
E The job started with a procedure start request.
J The job is a prestart job.
P The job is a print driver job.
T The job is a System/36 multiple requester terminal (MRT) job.
U Alternate spool user.

Job switches. The current setting of the job switches used by this job. This value is for jobs whose status is *ACTIVE. For jobs with a status of *OUTQ or *JOBQ, the value is blank.

Job type. The type of job. The possible values for this field are:

blank The job is not a valid job.
A The job is an autostart job.
B The job is a batch job.
I The job is an interactive job.
M The job is a subsystem monitor job.
R The job is a spooled reader job.
S The job is a system job.
W The job is a spooled writer job.
X The job is the SCPF system job.

Refer to "Comparing Job Type and Subtype with the Work with Active Job Command" on page 63-34 for information about how the job type field and the job subtype field equate to the type field in the Work with Active Job (WKRACTJOB) command.

Language ID. The language identifier associated with this job.

Logging level. What type of information is logged. The possible values are:

0 No messages are logged.
1 All messages sent to the job's external message queue with a severity greater than or equal to the message logging severity are logged.
2 The following information is logged:
- Level 1 information
- Requests or commands from CL programs for which the system issues messages with a severity code greater than or equal to the logging severity
- All messages associated with those requests or commands that have a severity code greater than or equal to the logging severity

3 The following information is logged:
- Level 1 information
- All requests or commands from CL programs
- All messages associated with those requests or commands that have a severity greater than or equal to the logging severity

4 The following information is logged:
- All requests or commands from CL programs
- All messages with a severity code greater than or equal to the logging severity

Logging of CL programs. Whether or not messages are logged for CL programs that are run. The possible values are *YES and *NO.

Logging severity. The minimum severity level that causes error messages to be logged in the job log.

Logging text. The level of message text that is written in the job log or displayed to the user when an error message is created according to the first two message logging values. The possible values are:

**MSG* Only the message is written to the job log.
**SECLVL* Both the message and the message help for the error message is written to the job log.
**NOLIST* If the job ends normally, there is no job log. If the job ends abnormally (if the job end code is 20 or higher), there is a job log. The messages appearing in the job's log contain both the message and the message help.

Maximum processing unit time. The maximum processing unit time (in milliseconds) that a routing step in this job can have to completely run the entire routing step. If the routing step is not finished before the maximum time is used up, running of the routing step is ended. A value of -1 is returned for *NOMAX. A zero is returned if the job is not active.

Maximum temporary storage. The maximum amount of auxiliary storage (in kilobytes) that a routing step in this job can use for processing. This temporary storage is used for storage required by the program itself and by implicitly created internal system objects used to support the routing step. (It is not storage in the QTEMP library.) If the maximum temporary storage is exceeded by a routing step, the routing step is ended. This does not apply to the use of permanent storage, which is controlled through the user profile. A value of -1 is returned for *NOMAX.

Mode name. The mode name of the advanced program-to-program communications device that started the job.

Number of auxiliary I/O requests. The number of auxiliary I/O requests for the job. This includes both database and nondatabase paging.

Number of bytes available. All of the available bytes for use in your application.

Note: When you request format JOBI0700, the actual length depends on how many libraries are in the library list.

Number of bytes returned. The number of bytes returned to the user. This may be some but not all of the bytes available.

Number of interactive transactions. The count of operator interactions, such as pressing the Enter key or a function key. This field is zero for jobs that have no interactions.

Number of libraries in SYSLIBL. The number of libraries in the system part of the library list.

Number of libraries in USRLIBL. The number of libraries in the user library list.

Number of product libraries. The number of product libraries found in the library list.

Output queue library name. The name of the library containing the output queue.

Output queue name. The name of the default output queue that is used for spooled output produced by this job. The default output queue is only for spooled printer files that specify *JOB for the output queue.

Output queue priority. The output priority for spooled output files that this job produces. The highest priority is 0, and the lowest is 9.

Print key format. Whether border and header information is provided when the Print key is pressed.

**NONE* The border and header information is not included with output from the Print key.
**PRTBDR* The border information is included with output from the Print key.
**PRTHDR* The header information is included with output from the Print key.
**PRTALL* The border and header information is included with output from the Print key.

Print text. The line of text (if any) that is printed at the bottom of each page of printed output for the job.

Printer device name. The printer device used for printing output from this job.

Processing unit used. The amount of processing unit time (in milliseconds) that the job used. A value of -1 is returned if the field is not large enough to hold the actual result. This has the same meaning as the +++ signs on the Work with Active Jobs display.

Product libraries if they exist. The library that contains product information. A blank is in the last position of the name.

Product return code. The return code set by the compiler for Integrated Language Environment (ILE) languages. Refer to the appropriate ILE-conforming language manual for possible values.

Program return code. If the job contains any RPG/400, COBOL, data file utility (DFU), or sort utility programs, the completion status of the last program that has finished running is shown; otherwise, a value of zero is shown.

Purge. Whether or not this job is eligible to be moved out of main storage and put into auxiliary storage at the end of a time slice or when it is beginning a long wait (such as waiting for a work station user's response). The possible values are:

**YES* The job is eligible to be moved out of main storage and put into auxiliary storage.
**NO* The job is not moved out of main storage. However, when some of main storage is needed to run other jobs in the same storage pool, pages belonging to this job may be moved (eight at a time) to auxiliary storage to accommodate pages needed by other jobs. When this job runs again, its pages are returned to main storage as they are needed.
blank Not used for job types *JOBQ or *OUTQ, or for invalid jobs.

Reserved. An ignored field.

Response time total. The total amount of response time for the job, in milliseconds. This value does not include the time used by the machine, attached input/output (I/O) hardware, and transmission lines for sending and receiving data. This field is zero for jobs that have no interactions. A value of -1 is returned if the field is not large enough to hold the actual result.

Run priority. The priority at which the job is currently running, relative to other jobs on the system. The run priority ranges from 1 (highest) to 99 (lowest).

Signed-on job. Whether the job is a to be treated like a signed-on user on the system.

0 The job should be treated like a signed-on user.
1 The job should not be treated like a signed-on user.

Sort sequence. The sort sequence table associated with this job.

Sort sequence library. The sort sequence library associated with this job.

Special environment. Whether the job is running in a particular environment. Possible values are:

**NONE* The job is not running in any special environment.
**S36* The job is running in the System/36 environment.

blank This job is not currently active.

Status message handling. Whether you want status messages displayed for this job. The possible values are:

**NONE* This job does not display status messages.
**NORMAL* This job displays status messages.

Status of job on the job queue. The status of this job on the job queue.

blank This job was not on a job queue.
SCD This job will run as scheduled.
HLD This job is being held on the job queue.
RLS This job is ready to be selected.

Submitter's job name. The job name of the submitter's job. If the job has no submitter, this field is blank.

Submitter's job number. The job number of the submitter's job. If the job has no submitter, this field is blank.

Submitter's message queue library name. The name of the library that contains the message queue. If the job has no submitter, this field is blank.

Submitter's message queue name. The name of the message queue where the system sends a completion message when a batch job ends. If the job has no submitter, this field is blank.

Submitter's user name. The user name of the submitter. If the job has no submitter, this field is blank.

Subsystem description library name. The library that contains the subsystem description. This value is only for jobs whose status is *ACTIVE. For jobs with a status of *OUTQ or *JOBQ, the value for this field is blank.

Subsystem description name. The name of the subsystem in which an active job is running. This value is only for jobs whose status is *ACTIVE. For jobs with status of *OUTQ or *JOBQ, the value for this field is blank.

System library list (for each library in the list). The system portion of the job's library list. A blank is in the last position of the name.

System pool identifier. The identifier of the system-related pool from which the job's main storage is allocated. These identifiers are not the same as those specified in the subsystem description, but are the same as the system pool identifiers shown on the system status display.

Temporary storage used. The amount of auxiliary storage (in kilobytes) that is currently allocated to this job.

Time separator. The value used to separate hours, minutes, and seconds when presenting a time.

Time slice. The maximum amount of processor time (in milliseconds) given to this job before other jobs are given the opportunity to run. The time slice establishes the amount of time needed by the job to accomplish a meaningful amount of processing. At the end of the time slice, the job might be put in an inactive state so that other jobs can become active in the storage pool.

Time-slice end pool. Whether you want interactive jobs moved to another main storage pool at the end of the time slice. The possible values are:

**NONE* The job does not move to another main storage pool when it reaches the end of the time slice.
**BASE* The job moves to the base pool when it reaches the end of the time slice.

Unit of work ID. The unit of work ID is used to track jobs across multiple systems. If a job is not associated with a source or target system using advanced program-to-program communications (APPC), this information is not used. Every job on the system is assigned a unit of work ID. The unit-of-work identifier is made up of:

Location name
CHAR(8). The name of the source system that originated the APPC job.
Network ID
CHAR(8). The network name associated with the unit of work.
Instance
CHAR(6). The value that further identifies the source of the job. This is shown as hexadecimal data.
Sequence number
CHAR(2). A value that identifies a checkpoint within the application program.

User library list (for each library in the list). The user portion of the job's library list. A blank is in the last position of the name.

User name. The user profile under which the job runs. The user name is the same as the user profile name and can come from several different sources depending on the type of job.

User return code. The user-defined return code set by ILE high-level language constructs. An example is the procedure return code in the C language.

Comparing Job Type and Subtype with the Work with Active Job Command

The following table compares the job type and job subtype fields returned by the QUSRJOBI API to the type field on the Work with Active Job (WRKACTJOB) command.

Figure 63-1 (Page 1 of 2). WRKACTJOB and QUSRJOBI API Comparison

Job Type Field	Job Type	Job Subtype
ASJ (Autostart)	A	blank
BCH (Batch)	B	blank

Figure 63-1 (Page 2 of 2). WRKACTJOB and QUSRJOBI API Comparison

Job Type Field	Job Type	Job Subtype
BCI (Batch immediate)	B	D
EVK (Started by a program start request)	B	E
INT (Interactive)	I	blank
MRT (Multiple requestor terminal)	B	T
PJ (Prestart job)	B	J
PDJ (Print driver job)	W	P
RDR (Reader)	R	blank
SYS (System)	S or X	blank
SBS (Subsystem monitor)	M	blank
WTR (Writer)	W	blank
blank (Alternative user subtype—not an active job)	B	U

Error Messages

CPF24B4 E Severe error addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C19 E Error with receiver variable.
CPF3C20 E Error found by program &1.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF3C36 E Number of parameters, &1, entered for this API was not valid.
CPF3C51 E Internal job identifier not valid.
CPF3C52 E Internal job identifier no longer valid.
CPF3C53 E Job &3/&2/&1 not found.
CPF3C54 E Job &3/&2/&1 currently not available.
CPF3C55 E Job &3/&2/&1 does not exist.
CPF3C57 E Not authorized to retrieve job information.
CPF3C58 E Job name specified is not valid.
CPF3C59 E Internal identifier is not blanks and job name is not *INT.
CPF9820 E Not authorized to use library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Job Queue Information (QSPRJOBQ) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Job queue name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Job Queue Information (QSPRJOBQ) API retrieves information associated with a specified job queue.

Authorities and Locks

Job Queue Library Authority
The caller needs *READ authority to the job queue library.

Job Queue Authority
The caller needs one of the following:

- *READ authority to the job queue.
- Job control special authority (*JOBCTL) if the job queue is operator controlled (OPRCTL(*YES)).
- Spool control special authority (*SPLCTL).

Job Queue Lock
This API gets an *EXCLRD lock on the job queue.

Required Parameter Group

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

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable provided by the receiver variable parameter. The amount of data returned is truncated if it is too small. A length of less than 8 is not valid.

Format name
INPUT; CHAR(8)
The content and format of the queue information being returned. The JOBQ0100 format must be used for the job queue information. See "JOBQ0100 Format" on page 63-36 to view the information returned for this format.

Job queue name
INPUT; CHAR(20)
The name of the job queue for which information is returned. The first 10 characters contain the queue name, and the second 10 characters contain the name of the library in which the queue resides.

The following special values are supported for the library name:

***LIBL** The library list used to locate the job queue.
***CURLIB** The current library for the job is used to locate the job queue.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

JOBQ0100 Format

The following table shows the information returned for the JOBQ0100 format. For more details about the fields in the following table see, "Field Descriptions" on page 63-36.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Job queue name
18	12	CHAR(10)	Job queue library name
28	1C	CHAR(10)	Operator controlled
38	26	CHAR(10)	Authority to check
48	30	BINARY(4)	Number of jobs
52	34	CHAR(10)	Job queue status
62	3E	CHAR(10)	Subsystem name
72	48	CHAR(50)	Text description

Field Descriptions

Authority to check. Whether the user must be the owner of the queue in order to control the queue by holding or releasing the queue. The possible values are:

**OWNER* Only the owner of the job queue can control the queue.
**DTAAUT* Any user with *READ, *ADD, or *DELETE authority to the job queue can control the queue.

Bytes available. Total format data length.

Bytes returned. Length of the data returned.

Job queue library name. The name of the library that contains the job queue.

Job queue name. The name of the job queue.

Job queue status. The status of the job queue. The status may be one of the following values:

RELEASED The queue is released.
HELD The queue is held.

Number of jobs. The number of jobs in the queue.

Operator controlled. Whether a user who has job control authority is allowed to control this job queue and manage the jobs on the queue.

**YES* Users with job control authority can control the queue and manage the jobs on the queue.
**NO* This queue and its jobs cannot be controlled by users with job control authority unless they also have other special authority.

Subsystem name. The name of the subsystem that can receive jobs from this job queue. If there is no name, then this queue is not associated with an active subsystem, and no job can be processed.

Text description. Text that briefly describes the job queue.

**BLANK* There is no text description of the job queue.

Error Messages

CPF2207 E Not authorized to use object &1 in library &3 type &2.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error occurred with receiver variable specified.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF3307 E Job queue &1 in &2 not found.
CPF3330 E Necessary resource not available.
CPF8121 E &8 damage on job queue &4 in &9.
CPF8122 E &8 damage on library &4.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Network Attributes (QWCRNETA) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Number of network attributes to retrieve	Input	Binary(4)
4	Network attribute names	Input	Array(*) of Char(10)
5	Error code	I/O	Char(*)

The Retrieve Network Attributes (QWCRNETA) API lets you retrieve network attributes.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. For the format, see "Format of Data Returned" on page 63-37.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable described in "Format of Data Returned" on page 63-37. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 28 bytes.

Number of network attributes to retrieve
INPUT; BINARY(4)
The total number of network attributes to be retrieved.

Network attribute names
INPUT: ARRAY(*) of CHAR(10)
The names of the network attributes to be retrieved. This can be a list of network attribute names where each name is 10 characters.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of Data Returned

The receiver variable holds the information returned about each network attribute.

The receiver variable has three logical parts:

1. The first field specifies the number of network attributes returned.
2. The next fields give the offsets to the network attributes returned. There is one offset field for each network attribute returned.
3. Next are the network attribute information tables for the network attributes returned. There is one network attribute information table for each network attribute.

The following table shows the format of the receiver variable. The offset fields are repeated until the offsets for all the network attributes returned are listed; the network attribute information table for each network attribute is repeated in the same way. For a detailed description of each field, see the "Field Descriptions."

The format of the receiver variable is:

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Number of network attributes returned
4	4	ARRAY(*) of BINARY(4)	Offset to network attribute information table
*	*	CHAR(*)	Network attribute information table
Note: Each network attribute in the table is represented by the standard network attribute information table described in "Network Attribute Information Table."			

To determine the length of the receiver variable, the following calculation should be done. For each network attribute to be returned, get the length of the data returned for the network attribute and add 24. After adding the values for each network attribute, add 4. This calculation takes into account the data alignment that needs to be done; therefore, this value is a worst-case estimate.

Network Attribute Information Table: The following table shows the format of the network attribute information table.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	CHAR(10)	Network attribute
10	A	CHAR(1)	Type of data
11	B	CHAR(1)	Information status
12	C	BINARY(4)	Length of data
16	10	CHAR(*)	Data

Field Descriptions

Data. The data returned for the network attribute.

Information status. Whether the information was available for the network attribute.

blank The information was available.
L The information was not available because the network attribute was locked.

Length of data. The length of the data returned for the network attribute. If this value is 0, the network attribute was not available.

Network attribute. The network attribute to be retrieved. See "Valid Network Attributes" for the list of valid network attributes.

Number of network attributes returned. The number of network attributes returned to the application.

Offset to network attribute information table. The offset from the beginning of the structure to the start of the network attribute information.

Type of data. The type of data returned.

blank The data was not available.
C The data is returned in character format.
B The data is returned in binary format.

Valid Network Attributes: For a detailed description of each field, see the "Network Attribute Field Descriptions."

Network attribute	Type	Description
ALRBCKFP	CHAR(16)	Alert backup focal point
ALRCTLD	CHAR(10)	Alert controller
ALRDFTFP	CHAR(10)	Alert focal point
ALRFTR	CHAR(20)	Alert filter
ALRHLDCNT	BINARY(4)	Alert hold count
ALRLOGSTS	CHAR(7)	Alert logging status
ALRPRIFP	CHAR(10)	Alert primary focal point

Network attribute	Type	Description
ALRRQSFP	CHAR(16)	Alert focal point to request
ALRSTS	CHAR(10)	Alert status
DDMACC	CHAR(20)	DDM request access
DFTCNNLST	CHAR(10)	PC default ISDN connection list
DFTMODE	CHAR(8)	Default mode
DFTNETTYPE	CHAR(10)	ISDN network type
DTACPR	BINARY(4)	Data compression
DTACPRINM	BINARY(4)	Intermediate data compression
JOBACN	CHAR(10)	Job action
LCLCPNAME	CHAR(8)	Local control point
LCLLOCNAME	CHAR(8)	Local location
LCLNETID	CHAR(8)	Local network ID
MAXINTSSN	BINARY(4)	Maximum sessions
MAXHOP	BINARY(4)	Maximum hop count
MSGQ	CHAR(20)	Message queue
NETSERVER	CHAR(85)	Server network ID
NODETYPE	CHAR(8)	APPN node type
OUTQ	CHAR(20)	Output queue
PNDSYSNAME	CHAR(8)	Pending system name
PCSACC	CHAR(20)	PC Support access
RAR	BINARY(4)	Addition resistance
SYSNAME	CHAR(8)	Current system name

Network Attribute Field Descriptions

Addition resistance. The Advanced Peer-to-Peer Networking (APPN) function routes addition resistance for an APPN node type of *NETNODE.

Alert backup focal point. Identifies the system that provides alert focal point services if the local system is unavailable and ALRPRIFP is *YES. The backup focal point is only used by systems in the primary sphere of control. The first eight characters are the control point name and the last eight characters are the network ID. *NONE means no backup focal point is defined.

Alert controller. The name of the controller to be used for alerts in a system service control point – physical unit (SSCP-PU) session. This controller is ignored if the system has a focal point (in which case the node is in another system's sphere of control). *NONE means no alert controller is defined.

Alert filter. The name of the filter object that is used by the alert manager when processing alerts. *NONE means no alert filter is being used. The first ten characters are the filter name and the last ten characters are the library name.

Alert focal point. Whether or not the system is an alert default focal point.

**NO* The system is not an alert default focal point.
**YES* The system is defined to be an alert default focal point and provides focal point services to all nodes in the network that are not being serviced by another focal point.

Alert focal point to request. Specifies the name of the system that is requested to provide focal point services. If a focal point is already defined for the entry point, it is taken away when the new focal point is requested. *NONE means no focal point is requested.

Alert hold count. The maximum number of alerts to be created before the alerts are sent over the System Service Control Point – Physical Unit (SSCP-PU) session. Alerts are held by the system until this number of alerts have been created. If the Alert Controller (ALRCTLD) network attribute is being used to send alerts (SSCP-PU session), alerts will be sent automatically regardless of the ALRHLDCNT network attribute when a switched connection is made for other reasons.

Alert logging status. Specifies which alerts are to be logged:

**LOCAL* Only locally created alerts are logged.
**RCV* Only Alerts received from other nodes are logged.
**ALL* Both locally created alerts and incoming alerts are logged.
**NONE* No alerts are logged.

Alert primary focal point. Whether or not the system is an alert primary focal point.

**NO* The system is not an alert primary focal point.
**YES* The system is defined to be an alert primary focal point and provides focal point services to all nodes in the network that are explicitly defined in the sphere of control.

Alert status. Alert status controls the creation of local alerts. The following is a list of values and their meanings:

**ON* Alerts are created by a system for all changeable conditions except "unattended" conditions.
**UNATTEND* Alerts are created by the system for all alert conditions including those which have the alert indicator in the message description set to *UNATTEND.
**OFF* Alerts are not created by the system.

APPN node type. The Advanced Peer-to-Peer Networking (APPN) node type can have the following values:

**ENDNODE* The node does not provide network services to other nodes, but may participate in the APPN network by using the services of an attached network server, or may operate

in a peer environment similar to migration end nodes.

**NETNODE* The node provides intermediate routing, route selection services, and distributed directory services for local users and to end nodes and migration end nodes that it is serving.

Current system name. The current system name that appears on displays.

Data compression. Whether data compression is used when the system is an SNA end node (the node containing either a primary or secondary LU). This field is used by mode descriptions that specify *NETATR for data compression. The following values are valid:

0 Data compression is not allowed on the session.

-1 Data compression is requested on the session by the local system. However, the request can be refused or changed to a lower compression level by the remote system. Data compression is allowed on the session if requested by the remote system.

-2 Data compression is allowed on the session by the local system if requested by a remote system. The local system does not request that compression be used.

-3 Data compression is required on the session. If the remote system does not change the levels of compression to the local system's exact requested levels, the session is not established. The data compression levels that the local system requires are the specified levels.

line speed If either the receiving or sending link has a line speed equal to or less than this specified line speed, this value indicates the need for compression by requesting that the remote systems compress the session data. Otherwise, this value does not indicate to the remote systems that there is a need to compress the data. Possible values range from 1 through 2 147 483 647 bits per second.

If data compression is requested by the remote system, the data compression levels used by the session are the lower of the requested levels and the configured levels (INDTACPR and OUTDTACPR).

DDM request access. The system processes distributed data management (DDM) requests from other systems as follows:

**REJECT* The system does not allow any DDM requests from remote systems. However, this system may use DDM to access files on remote systems.

**OBJAUT* All requests are allowed and controlled by the object authorization on the system.

program library The name of a user-written validation program that is called each time a DDM request is made from a remote system. This program indicates to DDM whether the request should proceed or be ended. System security still applies. The first ten characters are the program name and the last ten characters are the library name.

Default mode. The default mode name for the system.

Intermediate data compression. The level of data compression to request when the AS/400 system is an SNA intermediate node. The following are valid values:

0 Does not indicate to the remote systems that there is a need to compress the data.

-1 Indicates the need for compression by requesting that the remote systems compress the session data.

line speed If either the receiving or sending link has a line speed equal to or less than this specified line speed, this value indicates the need for compression by requesting that the remote systems compress the session data. Otherwise, this value does not indicate to the remote systems that there is a need to compress the data. Possible values range from 1 through 2 147 483 647 bits per second.

ISDN network type. The type of integrated services digital network (ISDN) to which the system is attached.

ATT5E42* This value is used when attaching to an ISDN in the United States or Canada that uses American Telephone and Telegraph 5ESS version 5E4.2 switching equipment.

**ATT5E5* This value is used when attaching to an ISDN in the United States or Canada that uses American Telephone and Telegraph 5ESS version 5E5 switching equipment.

**ATT5E6* This value is used when attaching to an ISDN in the United States or Canada that uses American Telephone and Telegraph 5ESS version 5E6 switching equipment.

**ATTG3* This value is used when attaching to an ISDN in the United States or Canada that uses AT&T Definity G1 switching equipment.

**BTNR191* This value is used when attaching to an ISDN in the United Kingdom that is controlled by British Telecom.

**CCITT88* The values recommended by the International Telegraph and Telephone Consultative Committee (CCITT) in 1988 are used.

**DBP1TR6* This value is used when attaching to an ISDN that is controlled by Germany's Post Telephone and Telegraph administration (PTT).

**ETSI* This value is used when attaching to an ISDN that uses the European Telecommunications Standards Institute (ETSI, also known as Euro-ISDN) standard.

*FTVN2 This value is used when attaching to Version 2 of the ISDN that is controlled by France's Post Telephone and Telegraph administration (PTT).

*INSNET64 This value is used when attaching to an ISDN that is controlled by Japan's Nippon Telephone and Telegraph Public Corporation.

*NISDN This value is used when attaching to an ISDN that uses the National ISDN-1 or the National ISDN-2 standard for North America.

*NT100B29 This value is used when attaching to an ISDN, in the United States or Canada, that uses Northern Telecom DMS100 Version BCS 29 switching equipment.

Job action. The action that is taken for any input stream received through the SNA distribution services (SNADS) network by the system.

*REJECT The input stream is rejected by the system. This action allows users to secure their system from input streams received through the network.

*FILE The input stream is filed in the queue of network files received for the user to whom it was sent. That user may then look at the input stream, end it, receive it, or submit it to a job queue.

*SEARCH The table of network job entries is searched to determine the action to be taken for the input stream.

Local control point. The local control point name for the system.

Local location. The default local location name for the system.

Local network ID. The local network ID assigned to the system.

Maximum hop count. The maximum number of times in an SNA distribution services (SNADS) network that a distribution queue entry originating at this node may be received and routed on the path to its final destination. If this number is exceeded, the distribution queue entry is ended. When the distribution queue entry is ended, a feedback status is sent back to the sender if it was requested.

Maximum sessions. The maximum number of advanced program-to-program communications (APPC) intermediate sessions for an Advanced Peer-to-Peer Networking (APPN) node type of *NETNODE.

Message queue. The name of the message queue to which messages received through the SNA distribution services (SNADS) network are sent for:

- Users who have no message queue specified in their user profile
- Users whose message queue is not available

The first 10 characters are the message queue name, and the last 10 characters are the library name.

Output queue. The name of the output queue to which spooled files received through the SNA distribution services (SNADS) network are sent for users whose output queue is not available. The first 10 characters are the output queue name, and the last 10 characters are the library name.

PC default ISDN connection list. The name of the default integrated services digital network (ISDN) connection list object.

PC Support access. The way in which the system processes PC Support requests from other systems.

*REJECT The system rejects every request from PC Support.

*OBJAUT Normal object authorizations are checked for the PC Support request. For example, authorization to retrieve data from a database file for a transfer function request is checked.

program library
The name of a user-written validation program that is called each time a PC Support application request comes from a personal computer. The program is passed two parameters: the first describes the PC request (the name of the application and type request); the second is used by the program to indicate to the PC Support application whether or not this PC request should be handled. The first 10 characters are the program name, and the last 10 characters are the library name.

Pending system name. The pending system name (if a change is pending). This will contain blanks if no change is pending.

Server network ID. The network node server of an Advanced Peer-to-Peer Networking (APPN) network (up to a maximum of five) for an APPN node type of *ENDNODE. The list is not used for an APPN node type of *NETNODE.

Error Messages

CPF1860 E Name in input list not valid.
CPF1861 E Error with receiver variable length.
CPF1862 E Number of values in input list not valid.
CPF24B4 E Severe error addressing parameter list.
CPF3C19 E Error with receiver variable.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Subsystem Information (QWDRSBSD) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Subsystem and library name	Input	Char(20)
5	Error code	I/O	Char(*)

The Retrieve Subsystem Information (QWDRSBSD) API retrieves information about a specific subsystem.

Authorities and Locks

Subsystem Description Authority
*USE

Library Authority *READ

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable to receive the subsystem information. This area must be large enough to accommodate the information specified.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. The length must be at least 8 bytes. If the variable is not long enough to hold the subsystem information, the data is truncated.

Format name
INPUT; CHAR(8)
The format of the subsystem information. You can use this format:

SBSI0100 Basic subsystem information. For details, see "SBSI0100 Format."

Subsystem and library name
INPUT; CHAR(20)
The subsystem about which to retrieve information, and the library in which the subsystem description is located. The first 10 characters contain the subsystem name, and the second 10 characters contain the library name. You can use these special values for the library name:

***CURLIB** The job's current library
***LIBL** The library list

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

SBSI0100 Format

The following table shows the information returned in the receiving variable for the SBSI0100 format. For a detailed description of each field, see "Field Descriptions."

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(10)	Subsystem description name
18	12	CHAR(10)	Subsystem description library name
28	1C	CHAR(10)	Subsystem status
38	26	CHAR(10)	Sign-on device file name
48	30	CHAR(10)	Sign-on device file library
58	3A	CHAR(10)	Secondary language library
68	44	BINARY(4)	Maximum active jobs
72	48	BINARY(4)	Currently active jobs
76	4C	BINARY(4)	Number of storage pools defined
Offsets vary. These five fields repeat, in the order listed, for each pool defined for the subsystem.		BINARY(4)	Pool ID
		CHAR(10)	Pool name
		CHAR(6)	Reserved
		BINARY(4)	Pool size
		BINARY(4)	Pool activity level

Field Descriptions

Bytes available. The total length of all data available.

Bytes returned. The length of the data actually returned. The number of bytes returned is always less than or equal to both the number of bytes available and the receiving variable length.

Currently active jobs. The number of jobs currently active in the subsystem. This number *includes* held jobs but *excludes* jobs that are disconnected or suspended because of a transfer secondary job or a transfer group job. If the subsystem status is *INACTIVE, this number is 0.

Maximum active jobs. The maximum number of jobs that can run or use resources in the subsystem at one time. If the subsystem description specifies *NOMAX, indicating that there is no maximum, this number is -1.

Number of storage pools defined. The number of storage pools defined for the subsystem. The maximum number of storage pools for a subsystem is currently 10. This number

determines how many times the pool ID, pool name, pool size, pool activity level, and reserved fields are repeated. Those five fields are repeated as a group for each pool defined for the subsystem.

Pool activity level. The maximum number of jobs that can be active in the pool at one time. If the pool name indicates a system-defined pool, the number returned is 0.

Pool ID. The pool ID for the subsystem pool.

Pool name. The name of the subsystem pool. If the pool is user-defined, the value of this field is *USERPOOL. If the pool is system-defined, the value is one of these names:

**BASE*
: The system base pool, which can be shared with other subsystems. The QBASPOOL system value defines the base pool's size. The QBASACTLVL system value defines its activity level.

**INTERACT*
: The shared pool used for interactive work.

**NOSTG*
: No storage size or activity level is assigned to this storage pool.

**SHRPOOL1–*SHRPOOL10*
: Shared pools.

**SPOOL*
: The shared pool for spooling writers.

The Change Shared Storage Pool (CHGSHRPOOL) command specifies the size and activity level of shared pools.

Pool size. If the pool name is *USERPOOL, the amount of storage, in kilobytes, that the pool attempts to allocate. If the pool has any other name, the value of this field is 0.

Reserved. An ignored field.

Secondary language library. The name of the subsystem's secondary language library.

Sign-on device file library. The name of the library in which the sign-on device file resides.

Sign-on device file name. The name of the sign-on file used by the subsystem.

Subsystem description library name. The name of the library in which the subsystem description resides.

Subsystem description name. The name of the subsystem about which information is being returned.

Subsystem status. The activity level of the subsystem. Valid values are *ACTIVE, indicating that the subsystem is running, and *INACTIVE, indicating that the subsystem is not running.

Error Messages

CPF1605 E Cannot allocate subsystem description &1.
CPF1606 E Error during allocation of subsystem &1.
CPF1607 E Previous request pending for subsystem &1.
CPF1608 E Subsystem description &1 not found.
CPF1619 E Subsystem description &1 in library &2 damaged.
CPF1835 E Not authorized to subsystem description.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF8122 E &8 damage on library &4.
CPF9807 E One or more libraries in library list deleted.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve System Status (QWCRSSTS) 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	Reset status statistics	Input	Char(10)
5	Error code	I/O	Char(*)

The Retrieve System Status (QWCRSSTS) API allows you to retrieve a group of statistics that represents the current status of the system.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that will receive the system status information being retrieved. For the format, see "Format of Data Returned" on page 63-43.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable described in "Format of Data Returned" on page 63-43. If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 8 bytes.

Format name
INPUT; CHAR(8)
The format of the information to be returned. You must use one of the following format names:

SSTS0100 Basic system status information about the signed-on users and batch jobs in the system. The information returned in this format is similar to the basic display of the Display System Status (DSPSYSSTS) command.

SSTS0200 System status information. The information returned in this format is similar to the disk information of the intermediate or advanced display of the DSPSYSSTS command.
SSTS0300 System status information. The information returned in this format is similar to the pool information of the intermediate or advanced display of the DSPSYSSTS command.

For more information about these formats, see "Format of Data Returned."

Reset status statistics
INPUT; CHAR(10)
Whether the status statistics are reset to zero, as if this were the first call to the API. The statistics will be reset before new information is gathered. This parameter will also reset the status statistics on the DSPSYSSTS and Work with System Status (WRKSYSSTS) commands. This parameter is ignored for format SSTS0100.

***YES** Statistics will be reset to zero.
***NO** Statistics will not be reset to zero.

Error code
I/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 Data Returned

The receiver variable holds the system status information returned.

SSTS0100 Format: The following table shows the information returned for the SSTS0100 format. For a detailed description of each field see "Field Descriptions" on page 63-44.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	CHAR(8)	Current date and time
16	10	CHAR(8)	System name
24	18	BINARY(4)	Users currently signed on
28	1C	BINARY(4)	Users temporarily signed off (disconnected)
32	20	BINARY(4)	Users suspended by system request
36	24	BINARY(4)	Users suspended by group jobs
40	28	BINARY(4)	Users signed off with printer output waiting to print
44	2C	BINARY(4)	Batch jobs waiting for messages
48	30	BINARY(4)	Batch jobs running
52	34	BINARY(4)	Batch jobs held while running
56	38	BINARY(4)	Batch jobs ending
60	3C	BINARY(4)	Batch jobs waiting to run or already scheduled
64	40	BINARY(4)	Batch jobs held on a job queue
68	44	BINARY(4)	Batch jobs on a held job queue
72	48	BINARY(4)	Batch jobs on an unassigned job queue
76	4C	BINARY(4)	Batch jobs ended with printer output waiting to print

SSTS0200 Format: The following table shows the information returned for the SSTS0200 format. For a detailed description of each field, see the "Field Descriptions" on page 63-44.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes available
4	4	BINARY(4)	Bytes returned
8	8	CHAR(8)	Current date and time
16	10	CHAR(8)	System name
24	18	CHAR(6)	Elapsed time
30	1E	CHAR(2)	Reserved
32	20	BINARY(4)	% processing unit used
36	24	BINARY(4)	Jobs in system
40	28	BINARY(4)	% permanent addresses
44	2C	BINARY(4)	% temporary addresses
48	30	BINARY(4)	System ASP
52	34	BINARY(4)	% system ASP used
56	38	BINARY(4)	Total auxiliary storage
60	3C	BINARY(4)	Current unprotected storage used
64	40	BINARY(4)	Maximum unprotected storage used

SSTS0300 Format: The following table shows the information returned for the SSTS0300 format. For a detailed description of each field, see the "Field Descriptions" on page 63-44.

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes available

Offset		Type	Field
Dec	Hex		
4	4	BINARY(4)	Bytes returned
8	8	CHAR(8)	Current date and time
16	10	CHAR(8)	System name
24	18	CHAR(6)	Elapsed time
30	1E	CHAR(2)	Reserved
32	20	BINARY(4)	Number of pools
36	24	BINARY(4)	Offset to pool information
40	28	BINARY(4)	Length of pool information entry
44	2C	CHAR(*)	Reserved
Offsets vary. These fields repeat, in the order listed, for each pool allocated by the system.		BINARY(4)	System pool
		BINARY(4)	Pool size
		BINARY(4)	Reserved size
		BINARY(4)	Maximum active jobs
		BINARY(4)	Database faults
		BINARY(4)	Database pages
		BINARY(4)	Nondatabase faults
		BINARY(4)	Nondatabase pages
		BINARY(4)	Active to wait
		BINARY(4)	Wait to ineligible
		BINARY(4)	Active to ineligible
		CHAR(10)	Pool name
		CHAR(10)	Subsystem name
		CHAR(10)	Subsystem library name
		CHAR(10)	Paging option

Field Descriptions

% permanent addresses. The percentage (in thousandths) of the maximum possible addresses for permanent objects that have been used. For example, a value of 41123 in binary would be 41.123.

% processing unit used. The average (in tenths) of the elapsed time during which the processing units were in use. For example, a value of 411 in binary would be 41.1.

% system ASP used. The percentage (in thousandths) of the system storage pool currently in use. When checksum protection is in effect, this percentage refers only to protected storage currently in use. Otherwise, it is the percentage of the total system storage pool currently in use. For example, a value of 41123 in binary would be 4.1123.

% temporary addresses. The percentage (in thousandths) of the maximum possible addresses for temporary objects that have been used. For example, a value of 41123 in binary would be 41.123.

Active to ineligible. The rate (in tenths), in transitions per minute, of transitions of jobs from an active condition to an ineligible condition. For example, a value of 123 in binary would be 12.3.

Active to wait. The rate (in tenths), in transitions per minute, of transitions of jobs from an active condition to a waiting condition. For example, a value of 123 in binary would be 12.3.

Batch jobs ended with printer output waiting to print. The number of completed batch jobs that produced printer output that is waiting to print.

Batch jobs ending. The number of batch jobs that are in the process of ending due to one of the following conditions:

- The job finishes processing normally.
- The job ends before its normal completion point and is being removed from the system.

Batch jobs held on a job queue. The number of batch jobs that were submitted, but were held before they could begin running.

Batch jobs held while running. The number of batch jobs that had started running, but are now held.

Batch jobs on a held job queue. The number of batch jobs on job queues that have been assigned to a subsystem, but are being held.

Batch jobs on an unassigned job queue. The number of batch jobs on job queues that have not been assigned to a subsystem.

Batch jobs running. The number of batch jobs currently running on the system.

Batch jobs waiting for messages. The number of batch jobs waiting for a reply to a message before they can continue to run.

Batch jobs waiting to run or already scheduled. The number of batch jobs on the system that are currently waiting to run, including those that were submitted to run at a future date and time. Jobs on the job schedule that have not been submitted are not included.

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 both the number of bytes available and the receiving variable length.

Current date and time. The date and time when the status was gathered. This is in system timestamp format.

Current unprotected storage used. The current amount of storage in use for temporary objects and machine data that are stored in unprotected storage when checksum protection

is in effect. This is used in conjunction with maximum unprotected storage to determine how much unprotected storage should be reserved when checksum protection is started. This value is in millions (M) of bytes.

Database faults. The rate (in tenths), shown in page faults per second, of database page faults against pages containing either database data or access paths. A page fault is a program notification that occurs when a page that is marked as not in main storage is referred to by an active program. An access path is the means by which the system provides a logical organization to the data in a database file. For example, a value of 123 in binary would be 12.3.

Database pages. The rate (in tenths), in pages per second, at which database pages are brought into the storage pool. A page is a 512-byte block of information that is transferable between auxiliary storage and main storage. For example, a value of 123 in binary would be 12.3.

Elapsed time. The time that has elapsed between the measurement start time and the current system time. This value is in the format HHMMSS where HH is the hour, MM is the minute, and SS is the second.

Jobs in system. The total number of user jobs and system jobs that are currently in the system. The total includes:

- All jobs on job queues waiting to be processed.
- All jobs currently active (being processed).
- All jobs that have completed running but still have output on output queues to be produced.

Length of pool information entry. The length of the information returned for each pool. If the receiver variable was not sufficiently large to hold all of the pool information, the amount of pool information returned may be less than this value.

Maximum active jobs. The maximum number of jobs that can be active in the pool at any one time.

Maximum unprotected storage used. The largest amount of storage (for temporary objects and machine data that are stored in unprotected storage when checksum protection is in effect) used at any one time since the last IPL. This value is in millions (M) of bytes.

Nondatabase faults. The rate (in tenths), in page faults per second, of nondatabase page faults against pages other than those designated as database pages. For example, a value of 123 in binary would be 12.3.

Nondatabase pages. The rate (in tenths), in pages per second, at which nondatabase pages are brought into the storage pool. For example, a value of 123 in binary would be 12.3.

Number of pools. The number of pools allocated when the information was gathered.

Offset to pool information. The offset from the beginning of the structure to the start of the pool information.

Paging option. Whether the system will dynamically adjust the paging characteristics of the storage pool for optimum performance. The following special values may be returned.

**FIXED*	The system does not dynamically adjust the paging characteristics.
**CALC*	The system dynamically adjusts the paging characteristics.
USRDFN	The system does not dynamically adjust the paging characteristics for the storage pool but uses values that have been defined through an API.

Pool name. The name of this storage pool. The name may be a number, in which case it is a private pool associated with a subsystem. The following special values may be returned.

**MACHINE*	The specified pool definition is defined to be the machine pool.
**BASE*	The specified pool definition is defined to be the base system pool, which can be shared with other subsystems.
**INTERACT*	The specified pool definition is defined to be the shared pool used for interactive work.
**SPOOL*	The specified pool definition is defined to be the shared pool used for spooled writers.
**SHRPOOL1–*SHRPOOL10*	The specified pool definition is defined to be a shared pool.

Pool size. The amount of main storage, in kilobytes, in the pool.

Reserved. An ignored field.

Reserved size. The amount of storage, in kilobytes, in the pool reserved for system use (for example, for save/restore operations). The system calculates this amount by using storage pool sizes and activity levels.

Subsystem library name. The library containing the subsystem description. This field will be blank for shared pools.

Subsystem name. The subsystem with which this storage pool is associated. This field will be blank for shared pools.

System ASP. The storage capacity of the system auxiliary storage pool. When checksum protection is in effect, this is the amount of space available for the storage of protected data only. Otherwise, this represents the amount of space available for storage of both protected and unprotected data. This value is in millions (M) of bytes.

System name. The name of the system where the statistics were collected.

System pool. The system-related pool identifier for each of the system storage pools that currently has main storage allocated to it.

Total auxiliary storage. The total auxiliary storage (in millions of bytes) on the system.

Users currently signed on. The number of users currently signed on the system. System request jobs and group jobs are not included in this number.

Users signed off with printer output waiting to print. The number of sessions that have ended with printer output files waiting to print.

Users suspended by group jobs. The number of user jobs that have been temporarily suspended by group jobs so that another job may be run.

Users suspended by system request. The number of user jobs that have been temporarily suspended by system request jobs so that another job may be run.

Users temporarily signed off (disconnected). The number of jobs that have been disconnected due to either the selection of option 80 (Temporary sign-off) or the entry of the Disconnect Job (DSCJOB) command.

Wait to ineligible. The rate (in tenths), in transitions per minute, of transitions of jobs from a waiting condition to an ineligible condition. For example, a value of 123 in binary would be 12.3.

Error Messages

CPF1E99 E Unexpected error occurred.
CPF1869 E Value not valid for reset status statistics.
CPF24B4 E Severe error addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error with receiver variable.
CPF3C21 E Format name not valid.
CPF3C24 E Receiver variable length not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve System Values (QWCRSVAL) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Number of system values to retrieve	Input	Binary(4)
4	System value names	Input	Array(*) of Char(10)
5	Error code	I/O	Char(*)

The Retrieve System Values (QWCRSVAL) API lets you retrieve system values.

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable that is to receive the information requested. For the format, see the "Format of Data Returned."

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable described in the "Format of Data Returned." If the length is larger than the size of the receiver variable, the results may not be predictable. The minimum length is 28 bytes.

Number of system values to retrieve
INPUT; BINARY(4)
The total number of system values to retrieve.

System value names
INPUT; ARRAY(*) of CHAR(10)
The names of the system values to be retrieved. This can be a list of system value names where each name is 10 characters.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Format of Data Returned

The receiver variable holds the information returned about each system value.

The receiver variable has three logical parts:

1. The first field specifies the number of system values returned.
2. The next fields give the offsets to the system values returned. There is one offset field for each system value returned.
3. Next are the system value information tables for the system values returned. There is one system value information table for each system value.

The following table shows the format of the receiver variable. The offset fields are repeated until the offsets for all the system values returned are listed; the system value information table for each system value is repeated in the same way. For a detailed description of each field, see the "Field Descriptions" on page 63-47.

The format of the receiver variable is:

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Number of system values returned
4	4	ARRAY(*) of BINARY(4)	Offset to system value information table

Offset		Type	Field
Dec	Hex		
*	*	CHAR(*)	System value information table. This field is repeated for each system value returned.

Note: Each system value in the table is represented by the standard system value information table described in "System Value Information Table" on page 63-47.

To determine the length of the receiver variable, the following calculation should be done. For each system value to be returned, get the length of the data returned for the system value and add 24. After adding the lengths for each system value, add 4. This calculation takes into account the data alignment that needs to be done; therefore, this value is a worst-case estimate.

System Value Information Table

The following table shows the format of the system value information table.

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	System value
10	A	CHAR(1)	Type of data
11	B	CHAR(1)	Information status
12	C	BINARY(4)	Length of data
16	10	CHAR(*)	Data

Field Descriptions

Data. The data returned for the system value.

Information status. Whether the information was available for the system value.

blank The information was available.
L The information was not available because the system value was locked.

Offset to system values information table. The offset from the beginning of the structure to the start of the system value information.

Length of data. The length of the data returned for the system value. If the information was not available, the length will be zero.

System value. The system value to be retrieved. See "Valid System Values" for the list of valid system values.

Number of system values returned. The number of system values returned to the application.

Type of data. The type of data returned.

C The data is returned in character format.
B The data is returned in binary format.
blank The data is not available.

Valid System Values

For a detailed description of each field, see the "System Value Field Descriptions" on page 63-49.

System value	Type	Description
QABNORMSW	CHAR(1)	Previous end of system indicator
QACGLVL	ARRAY(8) of CHAR(10)	Accounting level
QACTJOB	BINARY(4)	Active jobs
QADLACTJ	BINARY(4)	Additional active jobs
QADLSPLA	BINARY(4)	Additional storage
QADLTOTJ	BINARY(4)	Additional total jobs
QALWUSRDMN	ARRAY(50) of CHAR(10)	Allow user domain
QASTLVL	CHAR(10)	Assistance level
QATNPGM	CHAR(20)	Attention program
QAUDCTL	ARRAY(5) of CHAR(10)	Auditing control
QAUDENDACN	CHAR(10)	Auditing end action
QAUDFRCLVL	BINARY(4)	Auditing force level
QAUDLVL	ARRAY(16) of CHAR(10)	Auditing level
QAUTOCFG	CHAR(1)	Automatic configuration indicator
QAUTOVRT	BINARY(4)	Automatic configuration for virtual devices
QBASACTLVL	BINARY(4)	Base activity level
QBASPOOL	BINARY(4)	Base pool minimum size
QCCSID	BINARY(4)	Coded character set identifier
QCHRID	CHAR(20)	Character set and code page
QCMNRCYLMT	CHAR(20)	Communications recovery limit
QCNTRYID	CHAR(2)	Country identifier
QCONSOLE	CHAR(10)	Console name
QCRTAUT	CHAR(10)	Create authority
QCRTOBJAUD	CHAR(10)	Create object auditing
QCTLSBSD	CHAR(20)	Controlling subsystem
QCURSYM	CHAR(1)	Currency symbol
QDATE	CHAR(7)	System date
QDATFMT	CHAR(3)	Date format

System value	Type	Description
QDATSEP	CHAR(1)	Date separator
QDAY	CHAR(3)	Day
QDBRCVYWT	CHAR(1)	Database recovery wait
QDECFMT	CHAR(1)	Decimal format
QDEVNAMING	CHAR(10)	Device naming convention
QDEVRCYACN	CHAR(20)	Device recovery action
QDSCJOBITV	CHAR(10)	Disconnect job interval
QDSPSGNINF	CHAR(1)	Sign-on information
QHOUR	CHAR(2)	Hour
QHSTLOGSIZ	BINARY(4)	History log size
QIGC	CHAR(1)	DBCS installed
QIGCCDEFNT	CHAR(20)	Double-byte coded font name
QINACTITV	CHAR(10)	Inactive job time-out
QINACTMSGQ	CHAR(20)	Inactive message queue
QIPLDATTIM	CHAR(13)	Automatic IPL date and time
QIPLSTS	CHAR(1)	IPL status
QIPLTYPE	CHAR(1)	IPL type
QJOBMSGQFL	CHAR(10)	Job message queue full
QJOBMSGQMX	BINARY(4)	Job message queue maximum size
QJOBMSGQSZ	BINARY(4)	Job message queue initial size
QJOBMSGQTL	BINARY(4)	Maximum job message queue initial size
QJOBSPLA	BINARY(4)	Initial spooling size
QKBDBUF	CHAR(10)	Keyboard buffer
QKBDTYPE	CHAR(3)	Keyboard type
QLANGID	CHAR(3)	Language identifier
QLEAPADJ	BINARY(4)	Leap year adjustment
QLMTDEVSSN	CHAR(1)	Limit device session
QLMTSECOFR	CHAR(1)	Limit security officer
QMAXACTLVL	BINARY(4)	Maximum activity level
QMAXSGNACN	CHAR(1)	Maximum sign-on action
QMAXSIGN	CHAR(6)	Maximum not valid sign-on
QMCHPOOL	BINARY(4)	Machine pool size
QMINUTE	CHAR(2)	Minute
QMODEL	CHAR(4)	System model
QMONTH	CHAR(2)	Month
QPFRADJ	CHAR(1)	Performance adjustment
QPRBFTR	CHAR(20)	Problem filter
QPRBHLDITV	BINARY(4)	Problem hold interval
QPRTDEV	CHAR(10)	Printer device
QPRTKEYFMT	CHAR(10)	Print key format
QPRTTXT	CHAR(30)	Print text

System value	Type	Description
QPWDEXPITV	CHAR(6)	Days password valid
QPWDLMTAJC	CHAR(1)	Limit adjacent digits
QPWDLMTCHR	CHAR(10)	Limit characters
QPWDLMTREP	CHAR(1)	Limit repeat characters
QPWDMAXLEN	BINARY(4)	Maximum password length
QPWDMINLEN	BINARY(4)	Minimum password length
QPWDPOSDIF	CHAR(1)	Limit character positions
QPWDRQDDGT	CHAR(1)	Required password digits
QPWDRQDDIF	CHAR(1)	Duplicate password
QPWDVLDPGM	CHAR(20)	Password validation program
QPWRDWNLMT	BINARY(4)	Power down limit
QPWRRSTIPL	CHAR(1)	Power restore IPL
QRCLSPLSTG	CHAR(10)	Reclaim spool storage
QRMTIPL	CHAR(1)	Remote IPL
QRMTSIGN	CHAR(20)	Remote sign-on
QSCPFCONS	CHAR(1)	IPL action with console problem
QSECOND	CHAR(2)	Second
QSECURITY	CHAR(2)	Security level
QSFWERRLOG	CHAR(10)	Software error log
QSPCENV	CHAR(10)	Special environment
QSRLNBR	CHAR(8)	Serial number
QSRTSEQ	CHAR(20)	Sort sequence table
QSRVDMP	CHAR(10)	Service dump
QSTRPRTWTR	CHAR(1)	Start printer writer
QSTRUPPGM	CHAR(20)	Startup program name
QSTSMSG	CHAR(10)	Status messages
QSYSLIBL	ARRAY(15) of CHAR(10)	System library list
QTIME	CHAR(9)	System time
QTIMSEP	CHAR(1)	Time separator
QTOTJOB	BINARY(4)	Total jobs
QTSEPOOL	CHAR(10)	Time-slice end pool
QUPSDLYTIM	CHAR(20)	UPS delay time
QUPSMSGQ	CHAR(20)	UPS message queue
QUSRLIBL	ARRAY(25) of CHAR(10)	User library list
QUTCOFFSET	CHAR(5)	Coordinated universal time offset
QYEAR	CHAR(2)	Year

System Value Field Descriptions

Accounting level. QACGLVL is the accounting level. The possible values are:

**NONE* No accounting information is written to a journal.
**JOB* Job resource use is written to a journal.
**PRINT* The resources used for spooled and nonspooled print files are written to a journal.

Active jobs. QACTJOB is the initial number of active jobs for which auxiliary storage is to be allocated during IPL.

Additional active jobs. QADLACTJ specifies the additional number of active jobs for which auxiliary storage is to be allocated when the initial number of active jobs (the system value QACTJOB) is reached.

Additional storage. QADLSPLA specifies the additional storage to add to the spooling control block.

Additional total jobs. QADLTOTJ specifies the additional number of jobs for which auxiliary storage is to be allocated when the initial number of jobs (the system value QTOTJOB) is reached.

Allow user domain. QALWUSRDMN is the allow user domain system value. It specifies a list of library names that can contain user domain objects.

**ALL* All libraries on the system can contain user domain objects.
Library names A list of library names that can contain user domain objects.

Assistance level. QASTLVL is the assistance level system value. The value specifies the level of assistance available to users of the system.

**BASIC* Operational Assistant level of system displays is available.
**INTERMED* Intermediate level of system displays is available.
**ADVANCED* Advanced level of system displays is available.

Attention program. QATNPGM is the attention program system value. The first 10 characters contain the program name and the last 10 characters contain the library name. The following special values are allowed:

**ASSIST* The Operational Assistant main menu appears when the Attention key is pressed.
**NONE* No attention program is called when the Attention key is pressed.

Auditing control. The QAUDCTL system value is the on/off switch for object- and user-level auditing. The values allowed are:

**NONE* No auditing of objects and no auditing of user actions will be done on the system. In addition, no auditing that is controlled by the QAUDLVL system value will be done.
**OBJAUD* Objects that have been selected by the Change Object Auditing (CHGOBJAUD) command will be audited.
**AUDLVL* Changes controlled by the QAUDLVL system value and the AUDLVL parameter on the Change User Auditing (CHGUSRAUD) command will be audited.

Auditing end action. The QAUDENDACN system value indicates the action to be taken if auditing data cannot be written to the security auditing journal. These are the allowable values for the QAUDENDACN system value:

**NOTIFY* The action that caused the audit to be attempted will continue after notification of failure to send the journal entry to the security auditing journal is sent to the QSYSOPR and QSYSMSG message queues.
**PWRDWNSYS* The system ends with a system reference code (SRC) if sending of the audit data to the security audit journal fails. The system will then be brought up in a restricted state on the following IPL.

Auditing force level. The QAUDFRCLVL system value indicates to the system the number of auditing journal entries written to the security auditing journal before the auditing data is written to auxiliary storage. The following values are allowed:

0 The system will write the journal entries to auxiliary storage only when the system determines the journal entries should be written based on internal system processing.
1–100 The system will write the journal entries to auxiliary storage when this number of journal entries has been written to the security auditing journal.

Auditing level. QAUDLVL is the security auditing level. This system value specifies the level of security auditing that should occur on the system. The values allowed are:

**AUTFAIL* Authorization failures are audited.
**CREATE* The creation of objects is audited.
**DELETE* All object deletions are audited.
**JOBDTA* Actions by an audited user that affect a job will be audited.
**NONE* No auditing occurs on the system.
**OBJMGT* Function of generic objects is audited.
**OFCSRV* Auditing of OfficeVision/400 licensed program.
**PGMADP* Program adoption.
**PGMFAIL* Integrity violations (for example, blocked instruction, validation value failure, and domain violation) are audited.
**PRTDTA* Printing of spool files or direct printing.
**SAVRST* Save and restore information is audited.
**SECURITY* All security-related functions are audited.
**SERVICE* Use of the system service tools by a user will be audited.
**SPLFDTA* Spool file auditing.

| **SYSMGT* Use of system management functions by an audited user will be audited.

| **Automatic IPL date and time**. QIPLDATTIM is the system value for the date and time to automatically do an IPL of the system. It specifies a date and time when an automatic IPL should occur. The special value *NONE indicates that no timed automatic IPL is desired. The format of the field returned is CYYMMDDHHMMSS, where C is the century, YY is the year, MM is the month, DD is the day, HH is the hour, MM is the minute, and SS is the second.

| **Automatic configuration for virtual device**. QAUTOVRT is the system value for automatic configuration of virtual devices. This is a number in the range of 0 through 9999, which is the number of virtual devices that the user wants to have automatically configured.

| **Automatic configuration indicator**. The QAUTOCFG system value automatically configures devices. The value specifies whether devices that are added to the system are configured automatically.

| *0* Automatic configuration is off.
| *1* Automatic configuration is on.

| **Base activity level**. QBASACTLVL is the base-storage-pool activity level. This value indicates how many system and user jobs can compete at the same time for storage in the base storage pool.

| **Base pool minimum size**. QBASPOOL is the minimum size of the base storage pool. The base pool contains all main storage not allocated by other pools. QBASPOOL is specified in kilobytes.

| **Character set and code page**. QCHRID is the default character set and code page. The QCHRID system value is retrieved as a single character value; the first 10 characters contain the character set identifier right-justified. For example, the value 101 would be retrieved as 0000000101. The last 10 characters contain the code page identifier right-justified. For example, the value 37 would be retrieved as 0000000037.

| **Coded character set identifier**. QCCSID is the system value for coded character set identifiers.

| **Communications recovery limit**. QCMNRCYLMT is the system value for communications recovery limits. The QCMNRCYLMT system value is retrieved as a 20-character value; the first 10 characters contain the count limit right-justified. For example, the value 7 would be retrieved as 0000000007. The last 10 characters contain the time interval right-justified. For example, the value 117 would be retrieved as 0000000117.

| **Console name**. QCONSOLE is the console name. This value specifies the name of the display device that is the console.

| **Controlling subsystem**. QCTLSBSD is the controlling subsystem description. The controlling subsystem is the first subsystem to start after an IPL. The value of QCTLSBSD is a 20-character list of up to two 10-character values in which the first is the subsystem description name and the second is the library name.

| **Coordinated universal time offset**. QUTCOFFSET is the system value indicating the difference in hours and minutes between Universal Time Coordinated (UTC), also known as Greenwich mean time, and the current system (local) time.

| **Country identifier**. QCNTRYID is the system value for the country identifier. This value specifies the country identifier to be used as the default on the system.

| **Create authority**. QCRTAUT is the create authority system value. This value allows the default public authority for the create (CRTxxx) commands to be set system-wide. The values allowed are:

| **CHANGE* Allows you to change the contents of an object.
| **ALL* Allows you to read, change, delete, and manage the security of an object.
| **USE* Allows you to create an object, to display the contents of an object, or to refer to the contents of an attached object when a command being requested must access attached objects and their contents.
| **EXCLUDE* Allows no access to an object.

| **Create object auditing**. The QCRTOBJAUD system value indicates the default auditing value for new objects created into a library on the system. These are the allowable values for the QCRTOBJAUD system value.

| **NONE* No auditing entries are sent for this object when it is used or changed.
| **USRPRF* Auditing entries are sent for this object when it is used or changed by a user who is currently being audited. If the user who uses or changes this object is not being audited, no auditing entries are sent. To audit a user, you must use the Change User Auditing (CHGUSRAUD) command to change the user profile to that user profile.
| **CHANGE* Auditing entries are sent for this object when it is changed.
| **ALL* Auditing entries are sent for this object when it is used or changed.

| **Currency symbol**. QCURSYM is the system value for the currency symbol. QCURSYM can be any character except blank, hyphen (-), ampersand (&), asterisk (*), or zero (0).

| **Database recovery wait**. QDBRCVYWT is the database recovery wait indicator. QDBRCVYWT can be:

| *0* Does not wait for database recovery to complete before completing the IPL.
| *1* Waits for database recovery to complete before completing the IPL.

Date format. QDATFMT is the system date format. This system value can be YMD, MDY, DMY, or JUL (Julian format), where Y equals year, M equals month, and D equals day.

Date separator. QDATSEP is the character separator for dates. QDATSEP can be slash (/), hyphen (-), period (.), comma (,), or blank.

Day. QDAY is the system value for the day of the month or year (if the date format is Julian). For Julian dates only, QDAY is a 3-character value (001 through 366).

Days password valid. QPWDEXPITV is the system value for the password expiration interval. It controls the number of days that passwords are valid by keeping track of the number of days since you changed your password or created a user profile. The possible values are:

**NOMAX* A password can be used an unlimited number of days.
1–366 The number of days before the password cannot be used.

DBCS installed. QIGC is the DBCS version indicator. This value specifies if the DBCS version of the system is installed. QIGC can be:

0 A DBCS version is not installed.
1 A DBCS version is installed.

Decimal format. QDECFMT is the decimal format. QDECFMT must be one of the following characters:

blank Uses a period for a decimal point, a comma for a 3-digit grouping character, and zero-suppress to the left of the decimal point.
J Uses a comma for a decimal point and a period for a 3-digit grouping character. The zero-suppression character is in the second position (rather than the first) to the left of the decimal notation. Balances with zero values to the left of the comma are written with one leading zero (0,04). The J entry also overrides any edit codes that might suppress the leading zero.
I Uses a comma for a decimal point, a period for a 3-digit grouping character, and zero-suppress to the left of the decimal point.

Device naming convention. QDEVNAMING is the device naming convention. This value specifies what naming convention is used when the system automatically creates device descriptions. QDEVNAMING must be one of the following values:

**NORMAL* Naming conventions should follow AS/400 standards.
**S36* Naming conventions should follow System/36 standards.
**DEVADR* Device names are derived from the device address.

Device recovery action. QDEVRCYACN specifies what action to take when an I/O error occurs for an interactive job's work station. The values for QDEVRCYACN are:

**MSG*
Signals the I/O error message to the user's application program.
**DSCENDRQS*
Disconnects the job. When signing-on again, a cancel request function is performed to return control of the job back to the last request level.
**DSCMSG*
Disconnects the job. When signing-on again, an error message is sent to the user's application.
**ENDJOB*
Ends the job. A job log is produced for the job.
**ENDJOBNOLIST*
Ends the job. A job log is not produced for the job.

Disconnect job interval. QDSCJOBITV indicates the length of time, in minutes, an interactive job can be disconnected before it is ended. The values for QDSCJOBITV are:

5–1440 The range of the disconnect interval.
**NONE* There is no disconnect interval.

Double-byte coded font name. QIGCCDEFNT is the system value for the double-byte coded font name. QIGCCDEFNT is a 20-character list of up to two values in which the first 10 characters contain the coded font name and the last 10 characters contain the library name. *NONE means no coded font is identified to the system.

Duplicate password. QPWDRQDDIF controls duplicate passwords. It specifies whether the password must be different than the previous 32 passwords. The possible values are:

0 A password can be the same as any previously used password (except the immediately preceding password).
1 A password must be different from the previous 32 passwords.

History log size. QHSTLOGSIZ is the maximum number of records for each version of the history log.

Hour. QHOUR is the system value for the hour of the day. Hours are based on a 24-hour clock. Its value can range from 00 through 23.

Inactive job time-out. QINACTITV specifies the inactive job time-out interval in minutes. It specifies when the system takes action on inactive interactive jobs. QINACTITV must be one of the following values:

**NONE* The system does not check for inactive interactive jobs.
5–300 The number of minutes a job can be inactive before action is taken.

Inactive message queue. QINACTMSGQ is the system value for the inactive message queue. QINACTMSGQ is a

20-character list of up to two 10-character values where the first is the message queue name and the second is the library name. The following special values are allowed.

**DSCJOB* The interactive job is disconnected, as is any secondary or group job associated with it.
**ENDJOB* The interactive job is ended, along with any secondary job and any group job associated with it.

Initial spooling size. QJOBSPLA specifies the initial size of the spooling control block for a job.

IPL action with console problem. QSCPFCONS is the IPL action with a console problem indicator. This value specifies whether the IPL is to continue unattended or ends when the console is not operational when performing an attended IPL. QSCPFCONS can be:

0 End system.
1 Continue the IPL unattended.

IPL status. QIPLSTS is the IPL status indicator. This value indicates what form of IPL has occurred.

0 Operator panel IPL.
1 Automatic IPL after power restored.
2 Restart IPL.
3 Time-of-day IPL.
4 Remote IPL.

IPL type. QIPLTYPE indicates the type of IPL to perform. This value specifies the type of IPL performed when the system is powered on manually with the key in the normal position. QIPLTYPE can be:

0 Unattended.
1 Attended with dedicated service tools.
2 Attended with console in debug mode.

Job message queue full. QJOBMSGQFL specifies if the job message queue should be allowed to wrap.

**WRAP* Allow the job message queue to wrap.
**NOWRAP* Do not allow the job message queue to wrap.
**PRTWRAP* Print the message queue and then allow the message queue to wrap.

Job message queue initial size. QJOBMSGQSZ specifies the initial size of the job message queue. QJOBMSGQSZ is numeric and is specified in kilobytes.

Job message queue maximum size. QJOBMSGQMX specifies the maximum size of the job message queue. QJOBMSGQMX is numeric and is specified in megabytes.

Keyboard buffer. QKBDBUF specifies whether the type-ahead feature and Attention key buffering option should be used.

**TYPEAHEAD* The type-ahead feature is turned on, and the Attention key buffering option is turned off.
**NO* The type-ahead feature and the Attention key buffering option are turned off.
**YES* The type-ahead feature and the Attention key buffering option are turned on.

Keyboard type. QKBDTYPE specifies the language character set for the keyboard.

Language identifier. QLANGID is the system value for the language identifier. This system value specifies the language identifier to be used as the default for the system.

Leap year adjustment. QLEAPADJ is the system value for leap year adjustment. It is used to adjust the system calendar algorithm for the leap year in different calendar systems.

Limit adjacent digits. QPWDLMTAJC limits adjacent digits in a password. It specifies whether adjacent digits are allowed in passwords. The possible values are:

0 Adjacent digits are allowed in passwords.
1 Adjacent digits are not allowed in passwords.

Limit character positions. Limit password character positions. This system value controls the position of characters in a new password. This prevents the user from specifying the same character in a password corresponding to the same position in the previous password.

A change to this system value takes effect the next time a password is changed. The shipped value is 0.

0 The same characters can be used in a position corresponding to the same position in the previous password.
1 The same character cannot be used in a position corresponding to the same position in the previous password.

Limit characters. QPWDLMTCHR limits the use of certain characters in a password. The possible values are:

**NONE* There are no restricted characters.
restricted-characters Up to 10 restricted characters can be specified. Valid characters are A through Z, 0 through 9, and special characters such as number sign (#), dollar ($), underscore (_), or at sign (@).

Limit device session. QLMTDEVSSN is the system value for limiting device sessions. It controls whether a user can sign-on at more than one work station.

0 A user can sign-on at more than one device.
1 A user cannot sign-on at more than one device.

Limit repeat characters. QPWDLMTREP limits the use of repeating characters in a password. The possible values are:

0 Characters can be used more than once.
1 Characters cannot be used more than once.

Limit security officer. QLMTSECOFR is the system value for limiting QSECOFR device access. It controls whether users with *ALLOBJ or *SERVICE special authority need

explicit authority to specific work stations. The possible values are:

0 A user with *ALLOBJ or *SERVICE special authority can sign-on any device.
1 A user with *ALLOBJ or *SERVICE special authority can sign-on only at a device to which they have explicit authority.

Machine pool size. QMCHPOOL is the size of the machine storage pool. The machine storage pool contains shared machine and OS/400 licensed programs. QMCHPOOL is specified in kilobytes.

Maximum activity level. QMAXACTLVL is the maximum activity level of the system. This is the number of jobs that can compete at the same time for main storage and processor resources.

Maximum password length. QPWDMAXLEN specifies the maximum length of a password. It controls the maximum number of characters in a password. The possible values are:

1–10 The maximum number of characters that can be specified for a password.

Maximum job message queue initial size. QJOBMSGQTL is the maximum initial size of the job message queue. QJOBMSGQTL is numeric and is specified in kilobytes.

Maximum not valid sign-on. QMAXSIGN specifies the maximum number of incorrect sign-on attempts allowed. The possible values are:

1–25 The maximum number of sign-on attempts allowed.
**NOMAX* There is no maximum number of sign-on attempts.

Maximum sign-on action. QMAXSGNACN specifies the maximum sign-on attempts action or how the system reacts when the maximum number of consecutive incorrect sign-on attempts (the system value QMAXSIGN) is reached. The possible values are:

1 Varies off the device if limit is reached.
2 Disables the user profile if limit is reached.
3 Varies off the device and disables the user profile if the limit is reached.

Minimum password length. QPWDMINLEN specifies the minimum length of a password. It controls the minimum number of characters in a password. The possible values are:

1–10 The minimum number of characters that can be specified for a password.

Minimum problem retention. QPRBHLDITV allows you to specify the minimum number of days a problem is kept in the problem log. The range for this system value is 0 through 999 days.

Minute. QMINUTE is the system value for the minute of the hour. Its value can range from 00 through 59.

Month. QMONTH is the system value for the month of the year. It will be blank if the date format specified in system value QDATFMT is Julian (JUL). Its value can range from 1 through 12.

Password validation program. QPWDVLDPGM provides the ability for a user-written program to do additional validation on passwords. The first 10 characters contain the name of the program and the last 10 characters contain the library name. *NONE means a validation program is not used.

Performance adjustment. QPFRADJ indicates whether the system should adjust values during IPL and dynamically for system pool sizes and activity levels.

0 No performance adjustment.
1 Performance adjustment at IPL.
2 Performance adjustment at IPL and dynamically.
3 Dynamic performance adjustment.

Position characters. QPWDPOSDIF controls the position of characters in a new password. This prevents the user from specifying the same character in a password corresponding to the same position in the previous password. The possible values are:

0 The same characters can be used in a position corresponding to the same position in the previous password.
1 The same characters cannot be used in a position corresponding to the same position in the previous password.

Power down limit. QPWRDWNLMT is the maximum amount of time an immediate power down can take before processing is ended (abnormal end).

Power restore IPL. QPWRRSTIPL specifies whether the system should automatically do an IPL when utility power is restored after a power failure. The possible values are:

0 Automatic IPL is not allowed.
1 Automatic IPL is allowed.

Previous end of system indicator. QABNORMSW is the previous end of system indicator. The possible values are:

0 Previous end of system was normal.
1 Previous end of system was abnormal.

Print key format. QPRTKEYFMT specifies whether border and header information is provided when the Print key is pressed. The possible values are:

**NONE* The border and header information is not included with output from the Print key.
**PRTBDR* The border information is included with output from the Print key.
**PRTHDR* The header information is included with output from the Print key.

| **PRTALL* The border and header information is included with output from the Print key.

| **Print text**. QPRTTXT is the print text. This system value is used to print up to 30 characters of text on the bottom of listings and separator pages.

| **Printer device**. QPRTDEV is the default printer device description. This value specifies the default printer for the system.

| **Problem filter**. QPRBFTR specifies the name of the filter object that the service activity manager uses when processing problems. QPRBFTR is a 20-character list of up to two 10-character values in which the first value is the problem filter name and the second is the library name. *NONE means no problem filter is in use.

| **Problem hold interval**. QPRBHLDITV allows you to specify the minimum number of days a problem is kept in the problem log. After this time interval, the problem can be deleted using the Delete Problem (DLTPRB) command. The time interval starts as soon as it is put into the log.

| **Reclaim spool storage**. QRCLSPLSTG is reclaim spool storage system value. It allows for the automatic removal of empty spool database members. The values allowed are:

| **NOMAX* The maximum retention interval.
| **NONE* No retention interval.
| *1–366* Number of days empty spool database members are kept for new spooled file use.

| **Remote IPL**. QRMTIPL is the remote power on and IPL indicator. This value specifies if remote power on and IPL can be started over a telephone line. The possible values are:

| *0* Remote power on and IPL are not allowed.
| *1* Remote power on and IPL are allowed.

| **Remote sign-on**. QRMTSIGN specifies how the system handles remote sign-on requests. The user can specify a program and library to decide which remote sessions will be allowed and which user profiles can be automatically signed on from which locations. The first 10 characters contain the program name, and the last 10 characters contain the library name. QRMTSIGN can have the following values:

| **FRCSIGNON* All remote sign-on sessions are required to go through normal sign-on processing.
| **SAMEPRF* When the source and target user profile names are the same, the sign-on may be bypassed for remote sign-on attempts.
| **VERIFY* After verifying that the user has access to the system, the system allows the user to bypass the sign-on.
| **REJECT* No remote sign-on is allowed.

| **Required password digits**. QPWDRQDDGT specifies whether a digit is required in a new password. The possible values are:

| *0* A numeric digit is not required in new passwords.
| *1* A numeric digit is required in new passwords.

| **Second**. QSECOND is the system value for the second of the minute. Its value can range from 00 through 59.

| **Security level**. QSECURITY is the system security level indicator. The possible values are:

| *10* The system does not require a password to sign-on. The user has access to all system resources.
| *20* The system requires a password to sign-on. The user has access to all system resources.
| *30* The system requires a password to sign-on, and users must have authority to access objects and system resources.
| *40* The system requires a password to sign-on, and users must have authority to access objects and system resources. Programs that try to access objects through interfaces that are not supported will fail.
| *50* The system requires a password to sign-on, and users must have authority to access objects and system resources. Security and integrity of the QTEMP library and user domain (*USRxxx) objects are enforced. (Use system value QALWUSRDMN to change which libraries allow *USRxxx objects.) Programs fail if they try to pass unsupported parameter values to supported interfaces or if they try to access objects through interfaces that are not supported.

| **Note:** If this system value has been changed since the last IPL, this value is not the security level the system is currently using. This value will be in effect after the next IPL.

| **Serial number**. QSRLNBR is the system serial number. An example of a serial number is 1001003.

| **Service dump**. QSRVDMP specifies whether service dumps for unmonitored escape messages are created. The values that are allowed are:

| **DMPALLJOB* Service dumps will be created for all jobs.
| **DMPSYSJOB* Service dumps will be created for only system jobs, not user jobs.
| **DMPUSRJOB* Service dumps are created for only user jobs, not system jobs. System jobs include the system arbiter, subsystem monitors, LU services process, spool readers and writers, and the start-control-program-function (SCPF) job.
| **NONE* Do not request dumps in any jobs.

| **Sign-on information**. QDSPSGNINF is the system value for displaying sign-on information. The possible values are:

| *0* The sign-on information is not displayed.
| *1* The sign-on information is displayed.

| **Software error log**. QSFWERRLOG specifies whether software errors should be logged by the system. The allowed values are:

| **LOG* Software errors are logged.
| **NOLOG* No logging occurs.

Sort sequence table. QSRTSEQ is the name of the table used for the sort sequence. The first 10 characters contain the name of the table, and the last 10 characters contain the library name. The values for QSRTSEQ are:

**HEX* No sort sequence table is used. The hexadecimal values of the characters are used to determine the sort sequence.

**LANGIDSHR*
The sort sequence table used can contain the same weight for multiple characters. The shared weight sort table associated with the language specified in the LANGID parameter is used.

**LANGIDUNQ*
The sort sequence table used must contain a unique weight for each character in the code page, and it is the unique weight sort table associated with the language specified in the LANGID parameter.

sort sequence table name
The name and library of the sort sequence table to be used.

Special environment. QSPCENV specifies the system environment used as the default for all users. The possible values are:

**NONE* You enter the AS/400 environment when you sign-on.

**S36* You enter the System/36 environment when you sign-on.

Start printer writer. QSTRPRTWTR specifies whether printer writers are started at IPL. QSTRPRTWTR can be:

0 Do not start printer writers.

1 Start printer writers.

Startup program name. QSTRUPPGM is the startup program. This value specifies the name of the program called from an autostart job when the controlling subsystem is started. The first 10 characters contain the program name, and the last 10 characters contain the library name. *NONE means the autostart job ends normally without calling a program.

Status messages. QSTSMSG specifies whether or not the status messages are displayed. The values allowed are:

**NORMAL* Status messages are displayed.

**NONE* Status messages are not displayed.

System date. QDATE is the system date. QDATE is composed of the following system values: QYEAR, QMONTH, and QDAY. The format of the field returned is CYYMMDD where C is the century, YY is the year, MM is the month, and DD is the day.

System library list. QSYSLIBL is the system part of the library list. The list can contain as many as 15 names.

System model. QMODEL is the system model number. Examples of values you may see are B10, B20, B35, or B70.

System time. QTIME is the system value for the time of day. QTIME is composed of the following system values: QHOUR, QMINUTE, and QSECOND. QTIME has the format HHMMSSXXX, where HH equals hours, MM equals minutes, SS equals seconds, and XXX equals milliseconds.

Time separator. QTIMSEP is the character separator for time. QTIMSEP must be one of the following values: colon (:), period (.), comma (,), or blank.

Time-slice end pool. QTSEPOOL is the time-slice end pool. This value specifies whether interactive jobs should be moved to another main storage pool when they reach time-slice end. The values allowed are:

**NONE* Jobs are not moved to the base storage pool when time-slice end is reached.

**BASE* Jobs are moved to the base pool when time-slice end is reached.

Total jobs. QTOTJOB specifies the initial number of jobs for which auxiliary storage is allocated during IPL.

UPS delay time. The uninterruptible-power-supply (UPS) delay time specifies the amount of time that elapses before the system automatically powers down following a power failure. When a change in power activates the UPS, messages are sent to the UPS message queue (the system value QUPSMSGQ). This system value is only meaningful if your system has a battery power unit or has an uninterruptible power supply attached.

A change to this system value takes effect the next time there is a power failure. The shipped value is *CALC. The allowed values are:

**BASIC* Powers only the PRC, IOP cards, and Load Source direct-access storage device. The appropriate wait time, in seconds, is calculated. (This should be used only if you have the battery power unit or an uninterruptible power supply without every rack being connected.)

Note: All other values indicate that all racks have an uninterruptible power supply.

**CALC* Calculates the appropriate wait time.

**NOMAX* Starts no action.

0 Automatically powers down the system.

1–99999 Powers down the system after the specified number of seconds.

The QUPSDLYTIM system value is in the form of a two-item list. The first item is the value the user specified on the CHGSYSVAL command. The second item is the delay time, which is either what the user specified, or, if *CALC or *BASIC is specified, the calculated delay time.

UPS message queue. The QUPSMSGQ system value is the message queue that is to receive uninterruptible-power-supply messages. QUPSMSGQ is a 20-character list of up to two values in which the first 10 characters contain the message queue name, and the last 10 characters contain the library name.

User library list. QUSRLIBL is the default for the user part of the library list. The list can contain as many as 25 names.

Year. QYEAR is the system value that specifies the last 2 digits for the year. Its value can range from 0 through 99.

Error Messages

CPF1860 E Name in input list not valid.
CPF1861 E Error with receiver variable length.
CPF1862 E Number of values in input list not valid.
CPF24B4 E Severe error addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3C19 E Error with receiver variable.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Set Lock Flight Recorder (QWTSETLF) API

Parameters

Required Parameter:

1	Set value	Input	Char(4)

Optional Parameter:

2	Error code	I/O	Char(*)

The Set Lock Flight Recorder (QWTSETLF) API turns the lock flight recorder on and off. The value of *ON is passed to the program to turn the lock flight recorder on, and *OFF is passed to turn the lock flight recorder off.

When the lock flight recorder is turned on, the system will begin logging successful lock operations on devices in the lock flight recorder for the device being locked.

This API should be used only when recommended by your IBM service representative.

Required Parameter

Set value
INPUT; CHAR(4)
The value passed to turn the lock flight recorder on or off. The valid values are:

***ON** Turn flight recorder on.
***OFF** Turn flight recorder off.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF119B E Value &1 specified for parameter is not valid.
CPF3CF1 E Error code parameter not valid.
CPF8100 E All CPF81xx messages could be signaled. xx is from 01 to FF.
CPF9800 E All CPF98xx messages could be signaled. xx is from 01 to FF.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Set Trace (QWTSETTR) API

Parameters

Required Parameter Group:

1	Job Name	Input	Char(10)
2	User Name	Input	Char(10)

Optional Parameter:

3	Error code	I/O	Char(*)

The Set Trace (QWTSETTR) API starts a Trace Job (TRCJOB) command for the job passed on the parameter at the earliest point while the job is starting. This allows tracing of jobs early in the life of a job to help debug problems that could not have been done earlier because a user could not enter the command until the job was actually started.

The QWTSETTR API sets up the information about the job so that when that job is started a trace will begin.

The QWTSETTR API can be called multiple times to set up traces for multiple jobs. When the tracing is finished, the Control Trace (QWTCTLTR) API should be called using the *RESET value for the control value parameter to clear out all the job names. The Control Trace (QWTCTLTR) API must be called to turn on this trace activity.

The information set up by the QWTSETTR API will be in effect during an initial program load (IPL).

The information set up by the QWTSETTR API does not work for active jobs, but only for jobs that have not started yet.

If a job ends while the trace activity is running for that job, the trace information will be dumped to a spooled file.

The Trace Job (TRCJOB) command is issued in the job as if a user had typed in the command; therefore, the user (user name parameter) must have authorization to the TRCJOB command for this to work properly.

This API should only be used when recommended by an IBM service representative for collecting information for problems that occur early in job initiation.

Authorities and Locks

The user (user name parameter) must have authorization to the TRCJOB command for this to work properly.

Required Parameter Group

Job Name
INPUT; CHAR(10)
The name of the job that will be traced.

User Name
INPUT; CHAR(10)
The name of the user that will be traced.

Optional Parameter

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9. If this parameter is omitted, diagnostic and escape messages are issued to the application.

Error Messages

CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 25. Work Station Support APIs

Chapter 64. Work Station Support APIs

The work station support APIs let you use the type-ahead and attention key buffering functions in your applications. In the past, type-ahead and attention key buffering were active by default for all interactive jobs on the system.

Type-ahead, also called **keyboard buffering,** lets the user type data faster than it can be sent to the system. **Attention key buffering** determines how to process the action of pressing an Attention key. If attention key buffering is on, the Attention key is treated as any other key. If attention key buffering is not on, pressing the Attention key results in sending the information to the system even when other work station input is inhibited.

The work station support APIs and their functions are:

Query Keyboard Buffering (QWSQRYWS) determines the current type-ahead and attention key buffering settings.

Set Keyboard Buffering (QWSSETWS) controls the use of the type-ahead and attention key buffering functions.

You can enter the parameters for the QWSQRYWS and QWSSETWS APIs in mixed case. The APIs convert them to uppercase. For all other APIs, you must enter all parameters in uppercase.

The keyboard buffering data stream is supported by the following controllers:

ASCII Work Station Input/Output Processor
Twinaxial Work Station Input/Output Processor
5394 Remote Control Unit

IBM Personal Computer Systems attached via PC Support/400 or work station emulation (WSE) do not support the type-ahead data stream. Keyboard buffering for these devices is controlled through the emulation programs.

When a callable API parameter is not valid or the controller does not support type-ahead, the exception is signaled to the caller from the work station. All other exceptions, including device allocation and authority errors, are returned to the caller.

Query Keyboard Buffering (QWSQRYWS) API

Parameters

Required Parameter Group:

1	Keyboard buffering	Output	Char(1)

Optional Parameter:

2	Device name	Input	Char(10)

The Query Keyboard Buffering (QWSQRYWS) API sends the data stream command to query the current value for keyboard buffering for a specified display.

Required Parameter Group

Keyboard buffering
OUTPUT; CHAR(1)
The current setting for keyboard buffering for the device. Valid values are:

0 The type-ahead and attention key buffering functions are off.
1 The type-ahead and attention key buffering functions are on.
2 The type-ahead function is on, and the attention key buffering function is off.

Optional Parameter

Device name
INPUT; CHAR(10)
The display to query for the keyboard buffering value. You can specify a particular device or use this special value:

***REQUESTER** The job's requesting program device is queried for the keyboard buffering value. This is the default.

Error Messages

CPF94FC E Type-ahead data stream not supported by controller.
CPF94FD E Type-ahead option parameter value not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Set Keyboard Buffering (QWSSETWS) API

Parameters

Required Parameter Group:

1	Keyboard buffering	Input	Char(1)

Optional Parameter:

2	Device name	Input	Char(10)

The Set Keyboard Buffering (QWSSETWS) API controls the type-ahead and attention key buffering functions for a display. With the QWSSETWS API, you can:

- Turn both functions off
- Turn both functions on

- Turn on the type-ahead function without buffering the Attention key
- Send the data stream to a specific device

Any changes to the keyboard buffering value made through this program take effect immediately.

The keyboard buffering data stream is supported by the ASCII Work Station Input/Output Processor, Twinaxial Work Station Input/Output Processor, and 5394 Remote Control Unit. Keyboard buffering for personal computer systems attached via PC Support/400 or work station emulation (WSE) is controlled through the emulation programs; these devices do not support the type-ahead data stream.

Required Parameter Group

Keyboard buffering
INPUT; CHAR(1)
The setting for keyboard buffering for a display. Valid values are:

0 The type-ahead and attention key buffering functions are off.
1 The type-ahead and attention key buffering functions are on.
2 The type-ahead function is on, and the attention key buffering function is off.

Optional Parameter

Device name
INPUT; CHAR(10)
The device to set the keyboard buffering value on. You can specify the name of a particular device or use this special value:

***REQUESTER** The keyboard buffering value is set on the job's requesting program device. This is the default.

Error Messages

CPF94FC E Type-ahead data stream not supported by controller.
CPF94FD E Type-ahead option parameter value not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 26. Miscellaneous APIs

Chapter 65. Miscellaneous APIs

This chapter contains information about the following APIs:

Convert Date and Time Format (QWCCVTDT) allows you to convert date and time formats from one format to another format.

List Save File (QSRLSAVF) lists the contents of a save file.

Remove All Bookmarks from a Course (QEARMVBM) allows you to remove the bookmarks from a Tutorial System Support course.

Retrieve Main Storage (QVTRMSTG) allows you to get at main storage and retrieve the same type of information that you receive from the Start System Service Tools (STRSST) command.

The APIs in this chapter are presented in alphabetical order.

Convert Date and Time Format (QWCCVTDT) API

Parameters

Required Parameter Group:

1	Input format	Input	Char(10)
2	Input variable	Input	Char(*)
3	Output format	Input	Char(10)
4	Output variable	Output	Char(*)
5	Error code	I/O	Char(*)

The Convert Date and Time Format (QWCCVTDT) API converts date and time values from one format to another format. The QWCCVTDT API lets you:

- Convert a time-stamp (*DTS, for system time-stamp) value to character format
- Convert a character date-time value to time-stamp format
- Convert a date from one character format to another
- Retrieve the current machine clock time and return it in the format you specify

Required Parameter Group

Input format
INPUT; CHAR(10)
The format of the data you give QWCCVTDT to convert. Valid values are:

***CURRENT** Current machine clock time.

***DTS** System time-stamp. *DTS date values range only from August 23, 1928, 12:03:06.315 to May 10, 2071, 11:56:53.684. Converting a date outside this range to *DTS format results in a date within this range. When you convert a character date-time value to *DTS and back to character format, there is a rounding error of plus or minus 2 milliseconds.

***JOB** The format given in the DATFMT job attribute.

***SYSVAL** The format given in the QDATFMT system value.

***YMD** YYMMDD (year, month, day) format.

***MDY** MMDDYY (month, day, year) format.

***DMY** DDMMYY (day, month, year) format.

***JUL** Julian format (YYDDD (year, day of year)).

You can convert any format except *CURRENT to the same format without receiving an error. When you convert a format to the same format, the input variable is copied into the output variable without validation.

When you convert one character date format (that is, anything other than *CURRENT, the current machine-clock time, or *DTS, the system time-stamp) to another character date format, the date information is validated and converted. However, the time and milliseconds portions of the input variable are copied into the output variable without validation.

Input variable
INPUT; CHAR(*)
The data to be converted. See "Input Variable Format" on page 65-2 to determine the input variable.

Output format
INPUT; CHAR(10)
The format to convert the data to. Valid values are:

***DTS** System time-stamp
***JOB** The format given in the DATFMT job attribute
***SYSVAL** The format given in the QDATFMT system value
***YMD** YYMMDD format
***MDY** MMDDYY format
***DMY** DDMMYY format
***JUL** Julian format (YYDDD)

Output variable
OUTPUT; CHAR(*)
The converted data. If the output format is *DTS, the first 8 characters of this parameter are used. If the output format is one of the character formats (that is, anything other than *DTS), the first 16 characters of the output variable are used; for details, see "Character Date and Time Value Structure" on page 65-2.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Input Variable Format

This table shows the format used for the input variable parameter.

Input Format	Input Variable
*CURRENT	Parameter is ignored.
*DTS	System time-stamp. The first 8 characters are used.
All other character formats	The first 16 characters are used. See "Character Date and Time Value Structure."

Character Date and Time Value Structure

This table shows the structure used for the input and output variables:

Offset	Description
0	The century digit, which must be one of these: **0** 20th century **1** 21st century
1–6	The date, left-justified. This value cannot be all blanks or all zeros. Left-justify Julian dates, using blanks to fill the space.
7–12	The time, in HHMMSS (hours, minutes, seconds) format.
13–15	The milliseconds. This value cannot be blanks.

Error Messages

CPF1060 E Date not valid.
CPF1061 E Time not valid.
CPF1848 E Century digit &1 not valid.
CPF1849 E Milliseconds value &1 not valid.
CPF1850 E Format &1 not valid.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

List Save File (QSRLSAVF) API

Parameters

Required Parameter Group:

1	Qualified user space name	INPUT	CHAR(20)
2	Format name	INPUT	CHAR(8)
3	Qualified save file name	INPUT	CHAR(20)
4	Object name filter	INPUT	CHAR(10)
5	Object type filter	INPUT	CHAR(10)
6	Continuation handle	INPUT	CHAR(36)
7	Error code	I/O	CHAR(*)

The List Save File (QSRLSAVF) API lists the contents of a save file.[1] The generated list replaces any data that already exists in the user space; it does not add the new list to an existing one. The generated list is not sorted.

Authorities and Locks

Save File Library Authority	*READ
Save File Authority	*USE
Save File Lock	*EXCLRD
User Space Authority	*CHANGE
User Space Library Authority	*USE
User Space Lock	*EXCLRD

Required Parameter Group

Qualified user space 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 the save file. The possible format names are:

SAVF0100 Library level
SAVF0200 Object level
SAVF0300 Member level

For more information, see the specified formats in the "Format of the Generated List" on page 65-3.

Qualified save file name
INPUT; CHAR(20)
The save file about which to list information, and the library in which the save file is located. The first 10 characters contain the save file name, and the second 10 characters contain the library name. You can use these special values for the library name:

[1] A **save file** is a file allocated in auxiliary storage that can be used to store saved data on disk (without requiring diskettes or tapes), to do I/O operations from a high-level language program, or to receive objects sent through the network.

***CURLIB** The job's current library
***LIBL** The library list

Object name filter
INPUT; CHAR(10)
The name of the objects to search for. This name may be a simple name, a generic name, or the special value *ALL. If the name is not a valid name, an empty list will be returned. This field is ignored for the SAVF0100 format.

Object type filter
INPUT; CHAR(10)
The type of objects to search for. You may either enter a specific type or the special value *ALL. For a complete list of the available object types, see the *CL Reference*. This field is ignored for the SAVF0100 format and the SAVF0300 format.

Continuation handle
INPUT; CHAR(36)
The handle used to continue from a previous call to this API that resulted in partially complete information. You can determine if a previous call resulted in partially complete information by checking the information status field in the generic user space header following the API call. For information about the generic header, see the "User Space Format for List APIs" on page 2-7.

If the API is not attempting to continue from a previous call, this parameter must be set to blanks. Otherwise, a valid continuation value must be supplied. The value may be obtained from the continuation handle returned field in the header section. See "Format of the Generated List" for information about the header section.

Error code
I/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 save file list consists of:

- A user area
- A generic header
- An input parameter section
- A header section
- A list data section (containing one of the following) :
  - SAVF0100 format
  - SAVF0200 format
  - SAVF0300 format

For details about the user area and generic header, see the "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 65-4.

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 the DLTOLDSPLF example programs in Appendix A, "Examples."

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
28	1C	CHAR(10)	Save file name specified
38	26	CHAR(10)	Save file library name specified
48	30	CHAR(10)	Object name filter specified
58	3A	CHAR(10)	Object type filter specified
68	44	CHAR(36)	Continuation handle 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)	Save file name used
30	1E	CHAR(10)	Save file library name used
40	28	CHAR(36)	Continuation handle returned

SAVF0100 Format

Offset			
Dec	Hex	Type	Field
0	0	CHAR(10)	Library saved
10	A	CHAR(10)	Save command
20	14	CHAR(8)	Save date and time
28	1C	BINARY(4)	Auxiliary storage pool
32	20	BINARY(4)	Records
36	24	BINARY(4)	Objects saved
40	28	BINARY(4)	Access paths
44	2C	CHAR(10)	Save active
54	36	CHAR(6)	Release level
60	3C	CHAR(1)	Data compressed
61	3D	CHAR(8)	System serial number
69	45	CHAR(3)	Reserved

SAVF0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	A	CHAR(10)	Library saved
20	14	CHAR(10)	Object type
30	1E	CHAR(10)	Extended object attribute
40	28	CHAR(8)	Save date and time
48	30	BINARY(4)	Object size
52	34	BINARY(4)	Object size multiplier
56	38	BINARY(4)	Auxiliary storage pool
60	3C	CHAR(1)	Data saved
61	3D	CHAR(10)	Object owner
71	47	CHAR(20)	Document library object (DLO) name
91	5B	CHAR(63)	Folder
154	9A	CHAR(50)	Text description

SAVF0300 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	File name
10	A	CHAR(10)	Library saved
20	14	CHAR(10)	Member name
30	1E	CHAR(10)	Extended object attribute
40	28	CHAR(8)	Save date and time
48	30	BINARY(4)	Members saved

Field Descriptions

Access paths. The number of logical file access paths that were saved for the library.

Auxiliary storage pool. The auxiliary storage pool (ASP) of the object when it was saved. For the SAVF0100 format, this is the ASP of the library. For the SAVF0200 format, this is the ASP of the object. The possible values are:

1 System ASP
2 - 16 User ASPs

Continuation handle returned. A continuation point for the API.

This value is set based on the contents of the information status variable in the generic header for the user space. The following situations can occur:

- Information status–C. The information returned in the user space is valid and complete. No continuation is necessary and the continuation handle is set to blanks.
- Information status–P. The information returned in the user space is valid but incomplete. The user may call the API again, continuing where the last call ended. The continuation handle contains a value that may be supplied as an input parameter in later calls.
- Information status–I. The information returned in the user space is not valid or complete. The contents of the continuation handle are unpredictable.

Continuation handle specified. The handle used to continue from a previous call to this API that resulted in partially complete information.

Data compressed. Whether the data was stored in compressed format. The possible values are:

0 The data is not compressed.
1 The data is compressed.

Data saved. Whether the data for this object was saved with the object. The possible values are:

0 The data was not saved. The object's storage was freed by a previous save command before this save operation.
1 The data was saved. The object's storage was not freed by a previous save command before this save operation.

Document library object (DLO) name. The name of the document, folder, or mail object that was saved. If the object is a document or folder, the first 12 characters will contain the DLO name. If the object is a mail object, the full 20 characters will be used for the mail object name. If the save file does not contain DLO information, this field will be blank.

Extended object attribute. Extended information about the object type. If there is not an extended object attribute for the object, this field will be blank.

File name. The name of the file that was saved.

Folder. The name of the folder that was saved. The folder name is a fully qualified name. If the object is not a *FLR or *DOC object, this field will be blank. For *DOC and *FLR objects, this field will be set to the qualified name of the folder or to *NONE.

Format name. The format of the returned output.

Library saved. The name of the library from which the objects are saved.

Member name. The name of the file member that is saved. The member names are not in sorted order.

Members saved. The number of members saved for the file.

Object name. The name of the object saved. If the object is a DLO object, this field will contain the system name of the object.

Object name filter specified. The name of the objects to search for. Only objects with names that match the filter are listed.

Object owner. The name of the object owner's user profile.

Objects saved. The number of objects that are saved for this library.

Object size. The size of the object in units of the size multiplier. The true object size is equal to or smaller than the object size multiplied by the object size multiplier.

Object size multiplier. The value to multiply the object size by to get the true size. The value is 1 if the object is smaller than or equal to 999 999 999 bytes, and 1024 if it is larger.

Object type. The type of object. For a list of object types, see the *CL Reference*.

Object type filter specified. The type of objects to search for. Only object types that match the filter are listed.

Records. The number of records used to contain the saved information in the save file.

Release level. The earliest release level of the operating system on which the objects can be restored.

Reserved. An ignored field.

Save active. Whether objects in the library are allowed to be updated while they are being saved. The possible values are:

**LIB*	Objects in the library are saved while in use by another job. All of the objects in the library reached a checkpoint together and were saved in a consistent state in relationship to each other. All objects in the library are saved at the same time.
**NO*	Objects in the library are not saved while in use by another job.
**SYNCLIB*	Objects in the library are saved while in use by another job. All of the objects and all of the libraries in the save operation reached a checkpoint together. The objects and the libraries were saved in a consistent state in relationship to each other.
**SYSDFN*	Objects in the library are saved while in use by another job. Objects in the library may have reached a checkpoint at different times and may not be in a consistent state in relationship to each other.
**YES*	Document library objects are saved while in use by another job. This value is valid only if the SAVDLO command is used for the save operation.

Save command. The save command that is used when the save operation is performed. The possible values are:

QSYS	Contents of the save file are created by the operating system by using a function other than the CL commands.
SAVCFG	Saves configuration information.
SAVCHGOBJ	Saves objects that changed since the date and time specified on the referenced date parameter.
SAVDLO	Saves documents or folders located in library QDOC.
SAVLIB	Saves a copy of a library.
SAVOBJ	Saves an object or group of objects from the same library.
SAVSECDTA	Saves objects required for the security function.

Save date and time. The time at which the objects were saved in system time-stamp format.

Save file library name specified. The name of the save file library as specified in the call to the API.

Save file library name used. The name of the save file library used to produce the listing.

Save file name specified. The name of the save file as specified in the call to the API.

Save file name used. The name of the save file used to produce the listing.

System serial number. The serial number of the system on which the save was performed. If the save media is from a System/38, the system serial number will be blank.

Text description. The text description of the object. If the object is a DLO object, the following pertains:

- Characters 1 through 44 contain the text description.
- The last 6 characters are padded with blanks.

User space library name specified. The name of the library containing the user space as specified in the call to the API.

User space library name used. The name of the library used to produce the listing.

User space name specified. The name of the user space as specified in the call to the API.

User space name used. The name of the user space used to produce the listing.

Error Messages

CPF22FD E Continuation handle not valid for API &1.
CPF24B4 E Severe error while addressing parameter list.
CPF3704 E Request ended; data management error occurred.
CPD3723 D System may be too busy for save or restore operation.

	CPF4100 D	All CPF41xx messages could be returned. xx is from 01 to FF.
	CPF4200 D	All CPF42xx messages could be returned. xx is from 01 to FF.
	CPF4300 D	All CPF43xx messages could be returned. xx is from 01 to FF.
	CPF4500 D	All CPF45xx messages could be returned. xx is from 01 to FF.
	CPF4600 D	All CPF46xx messages could be returned. xx is from 01 to FF.
	CPF5100 D	All CPF51xx messages could be returned. xx is from 01 to FF.
	CPF5200 D	All CPF52xx messages could be returned. xx is from 01 to FF.
	CPF5300 D	All CPF53xx messages could be returned. xx is from 01 to FF.
CPF3743 E		File cannot be restored, displayed, or listed.
CPF3782 E		File &1 in &2 not a save file.
CPF3793 E		Machine storage limit reached.
CPF3812 E		Save file &1 in &2 in use.
CPF3C21 E		The format name specified is not valid.
CPF3CF1 E		Error code parameter not valid.
CPF8100 E		All CPF81xx messages could be returned. xx is from 01 to FF.
CPF9801 E		Object &2 in library &3 not found.
CPF9802 E		Not authorized to object &2 in &3.
CPF9803 E		Cannot allocate object &2 in library &3.
CPF9806 E		Cannot perform function for object &2 in library &3.
CPF9807 E		One or more libraries in library list deleted.
CPF9808 E		Cannot allocate one or more libraries on library list.
CPF9809 E		Library cannot be accessed.
CPF9810 E		Library &1 not found.
CPF9812 E		File &1 in library &2 not found.
CPF9820 E		Not authorized to use library &1.
CPF9822 E		Not authorized to file &1 in library &2.
CPF9830 E		Cannot assign library &1.
CPF9838 E		User profile storage limit exceeded.
CPF9872 E		Program &1 in library &2 ended. Reason code &3.

Remove All Bookmarks from a Course (QEARMVBM) API

Parameters

Required Parameter Group:

1	Course ID	Input	Char(10)
2	Error code	I/O	Char(*)

The Remove All Bookmarks from a Course (QEARMVBM) API removes all bookmarks from a Tutorial System Support course. This API provides support similar to option 9 (Remove bookmarks) on the Work with Courses display within the Start Education (STREDU) command.

Authority

The user must be an education administrator and have one of the following authorities:

Authority *ALLOBJ or *SECADM

Required Parameter Group

Course ID
INPUT; CHAR(10)
The ID of the course that is to have all bookmarks removed.

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

Error Messages

CPF1D50 E Not authorized to remove bookmarks.
CPF1D51 E Not all bookmarks removed.
CPF1D52 E Course not found.
CPF3CF1 E Error code parameter not valid.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Retrieve Main Storage (QVTRMSTG) API

Parameters

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Main storage address	Input	Char(12)
5	Main storage image selector	Input	Char(10)
6	Error code	I/O	Char(*)

The Retrieve Main Storage (QVTRMSTG) API retrieves information from main storage from either:

- The current system
- The space where main storage was dumped from a previous initial program load (IPL)

Authority

Authority *SERVICE

Required Parameter Group

Receiver variable
OUTPUT; CHAR(*)
The variable to receive a copy of the contents of the main storage. This area must be large enough to accommodate the specified information.

Length of receiver variable
INPUT; BINARY(4)
The length of the receiver variable. The length must be equal to or greater than 8 bytes. There is no limit to the variable length; however, if the length is not large enough to hold the copy of main storage, the data is truncated.

Format name
INPUT; CHAR(8)
The format of main storage being retrieved. Valid formats are:

STGI0100 Basic main storage information. See "STGI0100 Format."
STGI0200 Main storage information with tags. See "STGI0200 Format."

Main storage address
INPUT; CHAR(12)
The address of main storage being retrieved (a character representation of a hexadecimal address). If format STGI0200 is specified, the address must be on a 16-byte boundary.

Main storage image selector
INPUT; CHAR(10)
The main storage image from which to retrieve information. You can use these special values:

***MSTOR** The current main storage of the system
***DMPSPC** The main storage dump space

Error code
I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" on page 2-9.

STGI0100 Format

The following table shows the information returned in the receiving variable for the STGI0100 format. For a detailed description of each field, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	CHAR(*)	Main storage returned

STGI0200 Format

The following table shows the information returned in the receiving variable for the STGI0200 format. For a detailed description of each field, see "Field Descriptions."

Offset			
Dec	Hex	Type	Field
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available
8	8	BINARY(4)	Main storage length
12	12	BINARY(4)	Tag length
16	16	CHAR(*)	Main storage returned
*	*	CHAR(*)	Tag information returned

Field Descriptions

Bytes available. The total length of data available. For the QVTRMSTG API, the length will be equal to the bytes returned.

Bytes returned. The length of the data actually returned. The number of bytes returned is always less than or equal to both the number of bytes available and the receiving variable length.

Main storage length. The length of the main storage returned.

Main storage returned. A copy of the main storage from the address requested. The data is returned in hexadecimal format; 1 byte of main storage is represented as 1 byte in the returned data.

Tag information returned. A copy of the tag information for the main storage returned. To determine where the tag information returned starts, add the tag length and main storage length, plus 16.

Tag length. The length of tag information returned.

Error Messages

CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C5B E Option template entry is not valid.
CPF6FE0 E Requested main storage address is not valid.
CPF6FE1 E Selected main store image must be either *MSTOR or *DMPSPC.
CPF6FE2 E Retrieved data is not complete.
CPF6FE3 E Error occurred during data retrieval.
CPF6FE4 E Address specified must be on a 16-byte boundary.
CPF9872 E Program &1 in library &2 ended. Reason code &3.

Part 27. Reference Information

Appendix A. Examples

This appendix contains example programs that use APIs. Using the example programs, you can:

- Delete old spooled files using a program in one of the following languages:
  - RPG
  - COBOL
  - Pascal
  - System C/400 PRPQ

  Note: The programs and source code used as examples in the spooled file portion of this appendix exist only in printed form. They are not stored electronically on the AS/400 system.
- Change active jobs
- Change a job schedule entry
- Create your own telephone directory
- Create and manipulate a user index
- Create a batch machine
- Use the Create Program (QPRCRTPG) API
- Use profile handles
- Generate and send an alert
- Create diagnostic reports using message handling APIs
- List directories and subdirectories in spooled files
- Work with stream files
- Use the network management APIs
- Use the COBOL/400 APIs to control run units and error handling
- Use the user-defined communications APIs
- Use the Submit Debug Command API

This appendix also contains examples of exit programs to do the following:

- Use the Operational Assistant backup exit program
- Create the Program Temporary Fix exit program

Deleting Old Spooled Files

The following application program runs using the Delete Old Spooled Files (DLTOLDSPLF) command. This example has three major parts:

1. The DLTOLDSPLF command calls the delete old spooled files (DLTOLDSPLF) program in one of the following languages:
   - RPG
   - COBOL
   - Pascal
   - System C/400 PRPQ
2. The DLTOLDSPLF program is supplied in RPG, COBOL, Pascal, and System C/400 PRPQ. It does the following:
   a. Creates a user space (QUSCRTUS API).
   b. Generates a list of spooled files (QUSLSPL API).
   c. Retrieves information from a user space using one of the following:
      - QUSRTVUS API
      - QUSPTRUS API
   d. Retrieves more spooled file attribute information received from the user space (QUSRSPLA API).
   e. Calls the CLDLT program to delete the spooled files.
   f. Sends a message to the user (QMHSNDM API).
   g. Deletes the user space (QUSDLTUS API).
3. The CL delete (CLDLT) program does the following:
   a. Deletes the specified spooled files (DLTSPLF command).
   b. Sends a message if the spooled file was deleted (SNDPGMMSG command).

DLTOLDSPLF Command Source

The command source for the DLTOLDSPLF command follows:

```
/*************************************************************/
/*                                                           */
/* CMD:  DLTOLDSPLF                                          */
/*                                                           */
/* LANGUAGE:  CL COMMAND SOURCE                              */
/*                                                           */
/* DESCRIPTION:  COMMAND SOURCE FOR THE DLTOLDSPLF COMMAND WHICH*/
/*               INVOKES THE DLTOLDSPLF PROGRAM.             */
/*                                                           */
/*************************************************************/
     CMD PROMPT('DELETE OLD SPOOLED FILES')
/* PARAMETERS FOR LIST OF SPOOLED FILES (QUSLSPL)            */
     PARM KWD(USRPRFNME)  +
          TYPE(*SNAME)    +
          LEN(10)         +
          MIN(1)          +
          SPCVAL(*ALL)    +
          PROMPT('User Profile Name:')
     PARM KWD(OUTQUEUE)   +
          TYPE(QUAL1)     +
          MIN(1)          +
          PROMPT('Output Queue:')
/* INFORMATION NEEDED FOR PROGRAM                            */
     PARM KWD(DELETEDATE) +
          TYPE(*DATE)     +
          PROMPT('Last Deletion Date(MMDDYY):')
QUAL1:   QUAL TYPE(*NAME) LEN(10)  SPCVAL(*ALL)
     QUAL TYPE(*NAME) LEN(10) SPCVAL(*LIBL *CURLIB '         ') +
          PROMPT('Library Name:')
```

To create the CL command, specify the following:

```
CRTCMD CMD(QGPL/DLTOLDSPLF) PGM(QGPL/SPOOLINFO) +
    SRCFILE(QGPL/QCMDSRC) ALLOW(*IPGM *BPGM)
```

To delete old spooled files, you can use one of the application programs provided in the following languages:

- RPG
- COBOL
- Pascal
- System C/400 PRPQ

RPG DLTOLDSPLF Program: To delete old spooled files, use the following RPG program:

```
     H* ************************************************************
     H* ************************************************************
     H*                                                            *
     H* MODULE:    DLTOLDSPLF                                      *
     H*                                                            *
```

Examples: Deleting Old Spooled Files

```
     H* LANGUAGE:  RPG                                                *
     H*                                                               *
     H* FUNCTION:  THIS APPLICATION WILL DELETE OLD SPOOLED FILES     *
     H*            FROM THE SYSTEM, BASED ON THE INPUT PARAMETERS.    *
     H*                                                               *
     H* APIs USED:                                                    *
     H*            QUSCRTUS -- Create User Space                      *
     H*            QUSLSPLF -- List Spooled Files                     *
     H*            QUSRTVUS -- Retrieve User Space                    *
     H*            QUSRSPLA -- Retrieve Spooled File Attributes       *
     H*            QMHSNDPM -- Send Program Message                   *
     H*            QUSDLTUS -- Delete User Space                      *
     H*                                                               *
     H* ***************************************************************
     H* ***************************************************************
     I             'NUMBER OF SPOOLED -  C         MSGTXT
     I             'FILES DELETED:  '
     IMSGDTA      DS
     I                                       1  35 MSGDT1
     I                                      36  400DLTCNT
     ISTRUCT      DS
     I                                  B   1   40USSIZE
     I                                  B   5   80GENLEN
     I                                  B   9  120RTVLEN
     I                                  B  13  160STRPOS
     I                                  B  17  200RCVLEN
     I                                  B  21  240SPLF#
     I                                  B  25  280MSGDLN
     I                                  B  29  320MSGQ#
     I                                      33  38 FIL#
     I                                      39  42 MSGKEY
     I I           'DLTOLDSPLFQTEMP     '   43  62 USRSPC
     I I           '*REQUESTER          '   63  82 MSGQ
     IERRCOD      DS
     I                                  B   1   40BYTPRO
     I                                  B   5   80BYTAVA
     I                                      9  15 EXCID
     I                                     16  16 RSRVD
     I                                     17 116 EXCDTA
     ITGTDAT      DS
     I                                      1   1 TGTCEN
     I                                      2   3 TGTYR
     I                                      4   5 TGTMTH
     I                                      6   7 TGTDAY
     IGENHDR      DS
     I                                  B   1   40OFFSET
     I                                  B   9  120NUMENT
     I                                  B  13  160LSTSIZ
     IRTVVAR      DS
     I                                      1  10 USRNM1
     I                                     11  30 OUTQNA
     I                                     31  40 USRDT1
     I                                     41  50 FRMTY1
     I                                     51  66 IJOBID
     I                                     67  82 ISPLID
     IRCVVAR      DS
     I                                  B   1   40BYTRTN
     I                                  B   5   80BYTVAL
     I                                      9  24 JOBID
     I                                     25  40 SPLFID
     I                                     41  50 JOBNAM
     I                                     51  60 USRNME
     I                                     61  66 JOBNUM
     I                                     67  76 FILNAM
     I                                  B  77  800FILNUM
     I                                     81  90 FRMTYP
     I                                     91 100 USRDTA
     I                                    101 110 STATUS
     I                                    111 120 FILAVL
     I                                    121 130 HLDF
     I                                    131 140 SAVF
     I                                  B 141 1440TOTPAG
     I                                  B 145 1480PAGWRT
     I                                  B 149 1520STRPAG
     I                                  B 153 1560ENDPAG
     I                                  B 157 1600LASPAG
     I                                  B 161 1640RESPRT
     I                                  B 165 1680TOTCPY
     I                                  B 169 1720CPYLFT
     I                                  B 173 1760LPI
     I                                  B 177 1800CPI
     I                                    181 182 OUTPRI
     I                                    183 192 OUTQNM
     I                                    193 202 OUTQLB
     I                                    203 209 DATFOP
     I                                    203 203 DATCEN
     I                                    204 205 DATYR
     I                                    206 207 DATMTH
     I                                    208 209 DATDAY
     I                                    210 215 TIMFOP
     I                                    216 225 DEVFNA
     I                                    226 235 DEVFLB
     I                                    236 245 PGMOPF
     I                                    246 255 PGMOPL
     I                                    256 270 ACCCOD
     I                                    271 300 PRTTXT
     I                                  B 301 3040RCDLEN
     I                                  B 305 3080MAXRCD
     I                                    309 318 DEVCLS
     I                                    319 328 PRTTYP
     I                                    329 340 DOCNAM
     I                                    341 404 FLDNAM
     I                                    405 412 S36PRC
     I                                    413 422 PRTFID
     I                                    423 423 RPLUN
     I                                    424 424 RPLCHR
     I                                  B 425 4280PAGLEN
     I                                  B 429 4320PAGWID
     I                                  B 433 4360NUMSEP
     I                                  B 437 4400OVRLIN
     I                                    441 450 DBCSDA
     I                                    451 460 DBCSEC
     I                                    461 470 DBCSSO
     I                                    471 480 DBCSCR
     I                                  B 481 4840DBCSCI
     I                                    485 494 GRAPHI
     I                                    495 504 CODPAG
     I                                    505 514 FORNAM
     I                                    515 524 FORLIB
     I                                  B 525 5280SRCDRW
     I                                    529 538 PRTFON
     I                                    539 544 S36SPL
     I                                  B 545 5480PAGROT
     I                                  B 549 5520JUSTIF
     I                                    553 562 PRTBOT
     I                                    563 572 FLDRCD
     I                                    573 582 CTLCHR
     I                                    583 592 ALGFRM
     I                                    593 602 PRTQUA
     I                                    603 612 FRMFED
     I                                    613 683 VOLUME
     I                                    684 700 FLABID
     I                                    701 710 EXCTYP
     I                                    711 720 CHRCOD
     I                                  B 721 7240TOTRCD
     C* ***************************************************************
     C* ***************************************************************
     C*                                                                *
     C*              EXECUTABLE CODE STARTS HERE                       *
     C*                                                                *
     C* ***************************************************************
     C* ***************************************************************
     C*                                                                *
     C           *ENTRY    PLIST
     C                     PARM           USRNAM 10
     C                     PARM           OUTQ   20
     C                     PARM           DLTDAT  7
     C                     MOVE DLTDAT    TGTDAT
     C                     Z-ADD0         DLTCNT
     C                     MOVE *BLANKS   ERRCOD
     C                     Z-ADD0         BYTPRO
     C*                                                               *
     C* CREATE A USER SPACE TO STORE THE LIST OF SPOOLED FILES.       *
     C*                                                               *
     C                     CALL 'QUSCRTUS'
     C                     PARM           USRSPC
     C                     PARM *BLANKS   USEXAT 10
     C                     PARM 1024      USSIZE
     C                     PARM ' '       USINIT  1
     C                     PARM '*CHANGE 'USAUTH 10
     C                     PARM *BLANKS   USTEXT 50
     C                     PARM '*YES    'USREPL 10
     C                     PARM           ERRCOD
     C*                                                               *
     C* FILL THE USER SPACE JUST CREATED WITH SPOOLED FILES AS        *
     C* DEFINED IN THE CL COMMAND.                                    *
```

```
     C*                                                          *
     C                   CALL 'QUSLSPL'
     C                   PARM           USRSPC
     C                   PARM 'SPLF0100'FMTNM1  8
     C                   PARM           USRNAM
     C                   PARM           OUTQ
     C                   PARM '*ALL    'FRMTYP 10
     C                   PARM '*ALL    'USRDTA 10
     C                   PARM           ERRCOD
     C*                                                          *
     C* THE USER SPACE IS NOW FILLED WITH THE LIST OF SPOOLED FILES.  *
     C* NOW USE THE QUSRTVUS API TO FIND THE NUMBER OF ENTRIES AND    *
     C* THE OFFSET AND SIZE OF EACH ENTRY IN THE USER SPACE.          *
     C*                                                          *
     C                   Z-ADD16        GENLEN
     C                   Z-ADD125       STRPOS
     C*                                                          *
     C                   CALL 'QUSRTVUS'
     C                   PARM           USRSPC
     C                   PARM           STRPOS
     C                   PARM           GENLEN
     C                   PARM           GENHDR
     C                   PARM           ERRCOD
     C*                                                          *
     C* CHECK THE GENERIC HEADER DATA STRUCTURE FOR NUMBER OF LIST    *
     C* ENTRIES, OFFSET TO LIST ENTRIES, AND SIZE OF EACH LIST ENTRY. *
     C*                                                          *
     C                   Z-ADDOFFSET    STRPOS
     C                   ADD  1         STRPOS
     C                   Z-ADDLSTSIZ    RTVLEN
     C                   Z-ADD209       RCVLEN
     C                   Z-ADD1         COUNT  150
     C*                                                          *
     C* ************************************************************
     C* ************************************************************
     C*                                                          *
     C*     BEGINNING OF LOOP (DO WHILE COUNT <= NUMENT)         *
     C*                                                          *
     C* ************************************************************
     C*                                                          *
     C           COUNT     DOWLENUMENT
     C*                                                          *
     C* RETRIEVE THE INTERNAL JOB IDENTIFIER AND INTERNAL SPOOLED FILE*
     C* IDENTIFIER FROM THE ENTRY IN THE USER SPACE.  THIS INFORMATION*
     C* WILL BE USED TO RETRIEVE THE ATTRIBUTES OF THE SPOOLED FILE.  *
     C* THIS WILL BE DONE FOR EACH ENTRY IN THE USER SPACE.           *
     C*                                                          *
     C                   CALL 'QUSRTVUS'
     C                   PARM           USRSPC
     C                   PARM           STRPOS
     C                   PARM           RTVLEN
     C                   PARM           RTVVAR
     C                   PARM           ERRCOD
     C*                                                          *
     C* NOW RETRIEVE THE SPOOLED FILE ATTRIBUTES USING THE QUSRSPLA   *
     C* API.                                                      *
     C*                                                          *
     C                   MOVE *BLANKS   JOBINF
     C                   MOVEL'*INT'    JOBINF 26
     C                   MOVE IJOBID    JOBID
     C                   MOVE ISPLID    SPLFID
     C                   MOVEL'*INT'    SPLFNM 10
     C                   MOVE *BLANKS   SPLF#
     C*                                                          *
     C                   CALL 'QUSRSPLA'
     C                   PARM           RCVVAR
     C                   PARM           RCVLEN
     C                   PARM 'SPLA0100'FMTNM2  8
     C                   PARM           JOBINF
     C                   PARM           JOBID
     C                   PARM           SPLFID
     C                   PARM           SPLFNM
     C                   PARM           SPLF#
     C                   PARM           ERRCOD
     C*                                                          *
     C* CHECK RCVVAR DATA STRUCTURE FOR DATE FILE OPENED.             *
     C* DELETE SPOOLED FILES THAT ARE OLDER THAN THE TARGET DATE      *
     C* SPECIFIED ON THE COMMAND.  A MESSAGE IS SENT FOR EACH SPOOLED *
     C* FILE DELETED.                                             *
     C*                                                          *
     C*                                                          *
     C           DATYR     IFLT TGTYR
     C                     EXSR CLDLT
     C                     ELSE
     C           DATYR     IFEQ TGTYR
     C           DATMTH    IFLT TGTMTH
     C                     EXSR CLDLT
     C                     ELSE                                NOT LT MTH
     C           DATMTH    IFEQ TGTMTH
     C           DATDAY    IFLE TGTDAY
     C                     EXSR CLDLT
     C                     END                                 FOR LE DAY
     C                     END                                 FOR EQ MTH
     C                     END                                 FOR ELSE MTH
     C                     END                                 FOR EQ YR
     C                     END                                 FOR ELSE YR
     C*                                                                 *
     C* GO  BACK AND PROCESS THE REST OF THE ENTRIES IN THE USER      *
     C* SPACE.                                                         *
     C           LSTSIZ    ADD  STRPOS    STRPOS
     C           1         ADD  COUNT     COUNT
     C                     END
     C* ************************************************************ *
     C* ************************************************************ *
     C*                                                              *
     C*            END OF LOOP                                       *
     C*                                                              *
     C* ************************************************************ *
     C* ************************************************************ *
     C*                                                              *
     C* AFTER ALL SPOOLED FILES HAVE BEEN DELETED THAT MET THE        *
     C* REQUIREMENTS, SEND A FINAL MESSAGE TO THE USER.               *
     C* DELETE THE USER SPACE OBJECT THAT WAS CREATED.                *
     C*                                                              *
     C                   MOVELMSGTXT    MSGDT1
     C                   CALL 'QMHSNDM'
     C                   PARM *BLANKS   MSGID   7
     C                   PARM *BLANKS   MSGFIL 20
     C                   PARM           MSGDTA
     C                   PARM 40        MSGDLN
     C                   PARM '*INFO   'MSGTYP 10
     C                   PARM           MSGQ
     C                   PARM 1         MSGQ#
     C                   PARM *BLANKS   RPYMQ  10
     C                   PARM           MSGKEY
     C                   PARM           ERRCOD
     C*                                                                 *
     C* DELETE THE USER SPACE OBJECT THAT WAS CREATED.                 *
     C*                                                                 *
     C                   CALL 'QUSDLTUS'
     C                   PARM           USRSPC
     C                   PARM           ERRCOD
     C*                                                                 *
     C*                                                                 *
     C* ************************************************************ *
     C* ************************************************************ *
     C*                                                              *
     C*                      END OF PROGRAM                          *
     C*                                                              *
     C* ************************************************************ *
     C                   RETRN
     C*
     C* ************************************************************ *
     C*                                                              *
     C*                 CLDLT  SUBROUTINE                            *
     C*                                                              *
     C* THIS SUBROUTINE CALLS A CL PROGRAM THAT WILL DELETE A SPOOLED *
     C* FILE AND SEND A MESSAGE THAT THE SPOOLED FILE WAS DELETED.    *
     C*                                                              *
     C* ************************************************************ *
     C*                                                              *
     C           CLDLT     BEGSR
     C*                                                                 *
     C* KEEP A COUNTER OF HOW MANY SPOOLED FILES ARE DELETED.          *
     C*                                                                 *
     C                   ADD  1         DLTCNT
     C                   MOVE FILNUM    FIL#
     C                   CALL 'CLDLT'
     C                   PARM           FILNAM
     C                   PARM           JOBNUM
     C                   PARM           USRNME
     C                   PARM           JOBNAM
     C                   PARM           FIL#
     C                   PARM           FRMTYP
     C                   PARM           USRDTA
     C                   ENDSR
```

To create the RPG program, specify the following:

```
CRTRPGPGM PGM(QGPL/DLTOLDSPLF) SRCFILE(QGPL/QRPGSRC)
```

COBOL DLTOLDSPLF Program: To delete spooled files, you can use this COBOL DLTOLDSPLF program:

```
      *************************************************************
      *                                                           *
      * PROGRAM:  DLTOLDSPLF                                      *
      *                                                           *
      * LANGUAGE:  COBOL                                          *
      *                                                           *
      * DESCRIPTION:  DELETE OLD SPOOLED FILES                    *
      *                                                           *
      * APIs USED:  QUSCRTUS, QUSLSPL, QUSRTVUS, QUSRSPLA, QUSDLTUS,*
      *             AND QMHSNDM.                                  *
      *************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  DLTOLDSPLF.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-AS400.
       OBJECT-COMPUTER. IBM-AS400.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.

       DATA DIVISION.
       FILE SECTION.

       WORKING-STORAGE SECTION.
      *************************************************************
      * VALUES USED FOR THE QUSCRTUS PROGRAM                      *
      *************************************************************
       01  CRTUS-INFO.
           05  CRT-SPCNAME            PIC X(20)
                     VALUE "DLTOLDSPLFQTEMP     ".
           05  CRT-EXTATTR            PIC X(10) VALUE SPACE.
           05  CRT-SPCSIZE            PIC S9(9) BINARY VALUE 1024.
           05  CRT-INITSPACE          PIC X VALUE " ".
           05  CRT-AUTHORITY          PIC X(10) VALUE "*CHANGE   ".
           05  CRT-DESCRIPTION        PIC X(50) VALUE SPACE.
           05  CRT-USRRPL             PIC X(10) VALUE "*YES      ".

      *************************************************************
      * VALUES USED FOR THE ERROR CODE PARAMETER                  *
      *************************************************************
       01  ERRCODE-INFO.
           05  ERR-BYTPRO             PIC S9(9) BINARY VALUE 116.
           05  ERR-BYTAVA             PIC S9(9) BINARY VALUE ZERO.
           05  ERR-EXCID              PIC X(7) VALUE SPACES.
           05  ERR-RSRVD              PIC X(1) VALUE SPACE.
           05  ERR-EXCDTA             PIC X(100) VALUE SPACES.

      *************************************************************
      * VALUES USED FOR THE QUSRTVUS PROGRAM                      *
      *************************************************************
       01  RTV-START-POS              PIC S9(9) BINARY VALUE 1.
       01  RTV-LENGTH                 PIC S9(9) BINARY VALUE 140.
       01  RTV-HDR-INFO.
           05  FILLER PIC X(124).
           05  RTV-OFFSET-TO-ENTRIES  PIC S9(9) BINARY.
           05  FILLER PIC X(4).
           05  RTV-NUMBER-OF-ENTRIES  PIC S9(9) BINARY.
           05  RTV-SIZE-OF-EACH-ENTRY PIC S9(9) BINARY.

       01  LSTSPLA-INFO.
           05  LSTSPL-USER-NAME       PIC X(10).
           05  LSTSPL-OUTQ-NAME       PIC X(10).
           05  LSTSPL-OUTQLIB         PIC X(10).
           05  LSTSPL-USERDATA        PIC X(10).
           05  LSTSPL-FORMTYPE        PIC X(10).
           05  LSTSPL-INT-JOB-ID      PIC X(16).
           05  LSTSPL-INT-SPOOLID     PIC X(16).

       01  RTVSPLA-JOB-ID             PIC X(26) VALUE "*INT".

      *************************************************************
      * VALUES USED FOR THE QUSLSPL AND QUSRSPLA PROGRAM          *
      *************************************************************
       01  RSPLA-DATE.
           05 R-CENTURY               PIC X.
           05 R-MONTH                 PIC X(2).
           05 R-DAY                   PIC X(2).
           05 R-YEAR                  PIC X(2).
       01  LSPLA-FORMAT               PIC X(8)  VALUE "SPLF0100".
       01  LSPLA-USERDATA             PIC X(10) VALUE "*ALL      ".
       01  LSPLA-FORMTYPE             PIC X(10) VALUE "*ALL      ".
       01  RSPLA-JOB-NAME             PIC X(26) VALUE "*INT".
       01  RSPLA-NAME                 PIC X(10) VALUE "*INT".
       01  RSPLA-NUMBER               PIC S9(9) BINARY VALUE -1.
       01  RSPLA-FORMAT               PIC X(10) VALUE "SPLA0100  ".
       01  SPLA-VAR-LENGTH            PIC S9(9) BINARY VALUE 800.
       01  SPLA-VAR.
           05 FILLER                  PIC X(40).
           05 SPLA-JOBNAME            PIC X(10).
           05 SPLA-USERNAME           PIC X(10).
           05 SPLA-JOBNUMBER          PIC X(6).
           05 SPLA-FILENAME           PIC X(10).
           05 SPLA-FILENUMBER         PIC S9(9) BINARY.
           05 SPLA-FORMTYPE           PIC X(10).
           05 SPLA-USERDATA           PIC X(10).
           05 FILLER                  PIC X(102).
           05 OPENED-DATE             PIC X(7).
           05 FILLER                  PIC X(102).
           05 FILLER                  PIC X(593).
       01  DLT-COUNT                  PIC 9(15) VALUE 0.
       01  DLT-SPL-NUMBER             PIC 9(6).

      *************************************************************
      * VALUES USED FOR THE QMHSNDM PROGRAM                       *
      *************************************************************
        01  MSG-ID                    PIC X(7)  VALUE SPACE.
        01  MSG-FL-NAME               PIC X(20) VALUE SPACE.
        01  MSG-DATA.
            05  DATA-MD               PIC X(34)
                VALUE "NUMBER OF SPOOLED FILES DELETED : ".
            05  DLT-NUM-MD            PIC X(20) VALUE SPACE.
        01  MSG-DATA-LEN              PIC S9(9) BINARY VALUE 54.
        01  MSG-TYPE                  PIC X(10) VALUE "*INFO     ".
        01  MSG-QUEUE                 PIC X(20)
            VALUE "*REQUESTER          ".
        01  MSG-Q-NUM                 PIC S9(9) BINARY VALUE 1.
        01  RPY-MSG                   PIC X(10) VALUE SPACE.
        01  MSG-KEY                   PIC X(4)  VALUE SPACE.

      *************************************************************
      * PARAMETERS THAT ARE PASSED TO THIS PROGRAM FROM THE COMMAND *
      *************************************************************
       LINKAGE SECTION.
       01 PARM-USERNAME              PIC X(10).
       01 PARM-OUTQ                  PIC X(20).
       01 PARM-DATE.
          05 P-CENTURY               PIC X.
          05 P-MONTH                 PIC X(2).
          05 P-DAY                   PIC X(2).
          05 P-YEAR                  PIC X(2).

      *************************************************************
      * BEGINNING OF EXECUTABLE CODE.                             *
      *************************************************************

       PROCEDURE DIVISION USING PARM-USERNAME,
                                PARM-OUTQ,
                                PARM-DATE.

       MAIN-PROGRAM.

      *    *******************************************************
      *    * CREATE THE USER SPACE USING INPUT PARMS FOR THE CALL   *
      *    *******************************************************

           CALL "QUSCRTUS" USING CRT-SPCNAME,
                                 CRT-EXTATTR,
                                 CRT-SPCSIZE,
                                 CRT-INITSPACE,
                                 CRT-AUTHORITY,
                                 CRT-DESCRIPTION,
                                 CRT-USRRPL,
                                 ERRCODE-INFO.

      *    *******************************************************
      *    * LIST THE SPOOLED FILES TO THE USER SPACE OBJECT.       *
      *    *******************************************************
```

```
           CALL "QUSLSPL"  USING CRT-SPCNAME,
                                 LSPLA-FORMAT,
                                 PARM-USERNAME,
                                 PARM-OUTQ,
                                 LSPLA-FORMTYPE,
                                 LSPLA-USERDATA,
                                 ERRCODE-INFO.

      *    ************************************************************
      *    * RETRIEVE ENTRY INFORMATION FROM THE USER SPACE.         *
      *    ************************************************************

           CALL "QUSRTVUS" USING CRT-SPCNAME,
                                 RTV-START-POS,
                                 RTV-LENGTH,
                                 RTV-HDR-INFO,
                                 ERRCODE-INFO.

      *    ************************************************************
      *    * IF ANY SPOOLED FILES WERE FOUND MATCHING THE SEARCH     *
      *    * CRITERIA, RETRIEVE DETAILED INFORMATION AND DECIDE      *
      *    * WHETHER TO DELETE THE FILE OR NOT.                      *
      *    ************************************************************

           IF RTV-NUMBER-OF-ENTRIES IS GREATER THAN ZERO THEN
               ADD 1 TO RTV-OFFSET-TO-ENTRIES GIVING RTV-START-POS
               PERFORM CHECK-AND-DELETE THROUGH
                   CHECK-AND-DELETE-END RTV-NUMBER-OF-ENTRIES TIMES.

      *    ************************************************************
      *    * CALL THE QUSDLTUS API TO DELETE THE USER SPACE          *
      *    * WE CREATED, AND TO SEND A MESSAGE TELLING HOW MANY      *
      *    * SPOOLED FILES WERE DELETED.                             *
      *    ************************************************************

           CALL "QUSDLTUS" USING CRT-SPCNAME,
                                 ERRCODE-INFO.

           MOVE DLT-COUNT TO DLT-NUM-MD.
           CALL "QMHSNDM" USING MSG-ID,
                                MSG-FL-NAME,
                                MSG-DATA,
                                MSG-DATA-LEN,
                                MSG-TYPE,
                                MSG-QUEUE,
                                MSG-Q-NUM,
                                RPY-MSG,
                                MSG-KEY,
                                ERRCODE-INFO.

           STOP RUN.

      *    ************************************************************
      *    * CHECK THE DATE OF THE SPOOLED FILE.  IF IT IS OLDER     *
      *    * OR EQUAL TO THE DATE PASSED IN, CALL THE PROCEDURE      *
      *    * TO DELETE THE SPOOLED FILE.                             *
      *    ************************************************************

       CHECK-AND-DELETE.
           CALL "QUSRTVUS" USING CRT-SPCNAME,
                                 RTV-START-POS,
                                 RTV-SIZE-OF-EACH-ENTRY,
                                 LSTSPLA-INFO,
                                 ERRCODE-INFO.

      *    ************************************************************
      *    * ADVANCE TO NEXT SPOOLED FILE FOR PROCESSING THE CHECK   *
      *    * AND DELETE.                                             *
      *    ************************************************************

           ADD RTV-SIZE-OF-EACH-ENTRY TO RTV-START-POS GIVING
               RTV-START-POS.

      *    ************************************************************
      *    * RETRIEVE THE ATTRIBUTES FOR THE SPOOLED FILE TO GET     *
      *    * THE CREATE DATE FOR THE SPOOLED FILE.                   *
      *    ************************************************************

           CALL "QUSRSPLA" USING SPLA-VAR,
                                 SPLA-VAR-LENGTH,
                                 RSPLA-FORMAT,
                                 RSPLA-JOB-NAME,
                                 LSTSPL-INT-JOB-ID,
                                 LSTSPL-INT-SPOOLID,
                                 RSPLA-NAME,
                                 RSPLA-NUMBER,
                                 ERRCODE-INFO.

      *    ************************************************************
      *    * COMPARE THE CREATE DATE WITH THE DATE THAT WAS PASSED    *
      *    * IN AS PARAMETER.                                        *
      *    ************************************************************

           IF R-YEAR IS LESS THAN P-YEAR THEN
             PERFORM DLT-SPLF THROUGH DLT-SPLF-END
           ELSE
             IF R-YEAR IS EQUAL TO P-YEAR THEN
               IF R-MONTH IS LESS THAN P-MONTH THEN
                 PERFORM DLT-SPLF THROUGH DLT-SPLF-END
               ELSE
                 IF R-MONTH IS EQUAL TO P-MONTH THEN
                   IF R-DAY IS LESS THAN OR EQUAL TO P-DAY THEN
                     PERFORM DLT-SPLF THROUGH DLT-SPLF-END.

       CHECK-AND-DELETE-END.

      *    ************************************************************
      *    * THIS IS THE PROCEDURE TO DELETE THE SPOOLED FILE.       *
      *    * ALL OF THE SPOOLED FILES WITH CREATE DATE OLDER OR      *
      *    * EQUAL TO THE DATE PASSED IN AS PARAMETER WILL BE        *
      *    * DELETED.                                                *
      *    ************************************************************

       DLT-SPLF.
           ADD 1 TO DLT-COUNT.
           MOVE SPLA-FILENUMBER TO DLT-SPL-NUMBER.

           CALL "CLDLT" USING SPLA-FILENAME,
                              SPLA-JOBNUMBER,
                              SPLA-USERNAME,
                              SPLA-JOBNAME,
                              DLT-SPL-NUMBER,
                              SPLA-FORMTYPE,
                              SPLA-USERDATA.

       DLT-SPLF-END.
```

To create the COBOL program, specify the following:

```
CRTCBLPGM PGM(QGPL/DLTOLDSPLF) SRCFILE(QGPL/QCBLSRC)
```

Pascal DLTOLDSPLF Program: To delete spooled files, you can use this Pascal DLTOLDSPLF program:

```
(*****************************************************************)
(* PROGRAM:  DLTOLDSPLF                                          *)
(*                                                               *)
(* LANGUAGE:  PASCAL                                             *)
(*                                                               *)
(* DESCRIPTION:  THIS IS AN EXAMPLE PROGRAM FOR THE USE OF USER  *)
(*               SPACES WRITTEN IN AS/400 PASCAL.  THIS PROGRAM  *)
(*               WILL CREATE A USER SPACE AND LIST THE SPOOLED   *)
(*               FILES REQUESTED IN THE LIST ENTRY SECTION.  SOME*)
(*               DETAILED INFORMATION ABOUT THE SPOOLED FILE IS  *)
(*               RETRIEVED.  IF THE CREATION DATE OF THE FILE IS *)
(*               OLDER THAN THE COMPARISON DATE, THEN THE FILE IS*)
(*               DELETED.  AFTER ALL THE OLD SPOOLED FILES ARE   *)
(*               DELETED, THE USER SPACE IS DELETED.             *)
(*                                                               *)
(* APIs USED:  QUSCRTUS, QUSPTRUS, QUSLSPL, QUSRSPLA, QUSDLTUS,  *)
(*             QMHSNDM, QMHSNDPM                                 *)
(*                                                               *)
(*****************************************************************)
program dltoldsplf:

type
  sysname        = packed array (.1..10.) of char;
  qname          = packed array (.1..20.) of char;
  format         = packed array (.1..8.) of char;
  int_id         = packed array (.1..16.) of char;
  num_type       = packed array (.1..6.) of char;
  text_type      = packed array (.1..50.) of char;
  date_type      = packed array (.1..7.) of char;
  qjob_name_type = packed array (.1..26.) of char;
  rcv_var_type   = packed array (.1..724.) of char;
```

```
|   size_type      = packed array (.1..7.) of char;
|   header_space   = space (.500.) of integer;
|   entry_space    = space (.96.) of int_id;
|   error_code     = record
|                    bytes_prov   : integer;
|                    bytes_avail  : integer;
|                    except_id    : size_type;
|                    reserved     : string(1);
|                    except_data  : text_type
|                    end;

|   text_format    = record
|                    msg_text     : string(40);
|                    api_text     : string(10);
|                    end;

| var
| (* variables used into this program                              *)
|   us_name    :qname;     (* qualified user space name            *)
|   us_ext_attr:sysname;   (* extended attributes of user space    *)
|   us_init_val:char;      (* initial value of user space          *)
|   us_auth    :sysname;   (* authority of user space              *)
|   us_text    :text_type; (* text for user space                  *)

| (* variables passed into this program                            *)
|   user_name  :sysname;   (* user to delete spooled files from    *)
|   outq_name  :qname;     (* output queue to delete files from    *)
|   dlt_date   :date_type; (* cut-off date for deleted spooled files *)

| (* variables used for QUSCRTUS program                           *)
|   us_size_i  :integer;   (* integer value of user space size     *)
|   temp       :STRING(16);
|   replace    :sysname;

| (* variables used for QUSPTRUS program                           *)
|   us_header_ptr  :@header_space; (* pointer to the user space   *)

| (* variables used for QUSLSPL program                            *)
|   lspl_fmt   :format;    (* format of list entries from QUSLSPL  *)
|   form_type  :sysname;   (* form type of spooled files to delete *)
|   user_data  :sysname;   (* user data of spooled files to delete *)

| (* variables used for QUSRSPLA program                           *)
|   rcv_var    :rcv_var_type;    (* returned information          *)
|   rcv_var_len:integer;         (* length of returned variable   *)
|   rspla_fmt  :format;          (* format of info returned       *)
|   qjob_name  :qjob_name_type;  (* qualified job name, set to *INT *)
|   int_job_id :int_id;          (* internal job identifier       *)
|   int_spl_id :int_id;          (* internal spooled file identifier *)
|   splf_name  :sysname;         (* name of spooled file          *)
|   splf_number:integer;         (* number of spooled file        *)

| (* variables used for CLDLT program                             *)
|   d_splf     :sysname;         (*spooled file to delete         *)
|   d_job_num  :num_type;        (* job number for spooled file   *)
|   d_user_name:sysname;         (* user name for spooled file    *)
|   d_job_name :sysname;         (* job name for spooled file     *)
|   d_splf_num :num_type;        (* number of spooled file to delete*)
|   d_form_type:sysname;         (* spooled file form type        *)
|   d_user_data:sysname;         (* spooled file user data        *)

| (* declare pointers to receive input parameters since PARM       *)
| (* function returns type POINTER                                *)

|   sysname_ptr :@sysname;
|   qname_ptr   :@qname;
|   date_ptr    :@date_type;

| (* information retrieved from user space header after QUSLSPL    *)
|   list_offset     :integer;    (* decimal offset to list beginning *)
|   num_of_entries  :integer;    (* number of entries in list     *)
|   size_of_entries :integer;    (* size of each entry in list    *)
|   list_entry_ptr  :@entry_space; (* pointer to a list entry     *)

| (* some other variables used in the program                      *)
|   count          :integer;    (* loop counter                  *)
|   dlt_count_i     :integer;    (* integer value for delete count *)
|   rtv_date        :date_type;  (* date of retrieved spooled file *)
|   i_ptr           :@integer;   (* used for a conversion routine *)
| (* variables used for QMHSNDPM and QMHSNDM programs             *)
|   msg_id          :size_type;  (* identifying code for messages *)
|   msg_file        :qname;      (* name and library of message file *)
|   msg_length      :integer;    (* length of the predefined message *)
|   msg_type        :sysname;    (* type of message               *)
|   msg_queue       :qname;      (* list of message queue names   *)
|   pgm_queue       :sysname;    (* msg queue names for QMHSNDPM  *)
|   stack_cnt       :integer;    (* program stack counter         *)
|   msg_key         :num_type;   (* key to the message being sent *)
|   data_msg        :string(45); (* message text                  *)
|   rpy_msgq        :sysname;    (* name of reply message queue   *)
|   msg_que_num     :integer;    (* number of message queue in the *)
|                                 (* list passed                  *)

| (* Miscellaneous variables                                       *)
|   err_code        :error_code;
|   api_code        :integer;
|   msg_data        :text_format;

| (* declare external procedures                                   *)
| (* create a user space                                          *)
| procedure QUSCRTUS (VAR us_name    :qname;
|                     VAR ext_attr   :sysname;
|                     VAR init_size  :integer;
|                     VAR init_val   :char;
|                     VAR us_auth    :sysname;
|                     VAR text       :text_type;
|                     VAR replace    :sysname;
|                     VAR err_code   :error_code);nonpascal;

| (* get a pointer to a user space                                 *)
| procedure QUSPTRUS (VAR us_name  :qname;
|                     VAR us_ptr   :POINTER);nonpascal;

| (* API to delete user space                                      *)
| procedure QUSDLTUS (VAR us_name  :qname;
|                     VAR err_code :error_code);nonpascal;


| (* list spooled files in a user space                           *)
| procedure QUSLSPL (VAR us_name   :qname;
|                     VAR fmt       :format;
|                     VAR usr       :sysname;
|                     VAR outq      :qname;
|                     VAR form      :sysname;
|                     VAR usr_data  :sysname;
|                     VAR err_code  :error_code);nonpascal;

| (* retrieve detailed information of a spooled file              *)
| procedure QUSRSPLA (VAR rcv_var    :rcv_var_type;
|                     VAR rcv_len    :integer;
|                     VAR fmt        :format;
|                     VAR q_job_name :qjob_name_type;
|                     VAR int_job_id :int_id;
|                     VAR int_spl_id :int_id;
|                     VAR splf_name  :sysname;
|                     VAR splf_num   :integer;
|                     VAR err_code   :error_code);nonpascal;

| (* delete a spooled file                                        *)
| procedure CLDLT (VAR splf_name :sysname;
|                   VAR job_num   :num_type;
|                   VAR user_name :sysname;
|                   VAR job_name  :sysname;
|                   VAR splf_num  :num_type;
|                   VAR fm_type   :sysname;
|                   VAR usr_data  :sysname);nonpascal;

| (* Send message to a nonprogram message queue                    *)
| procedure QMHSNDM (VAR msg_id     :size_type;
|                     VAR msg_file   :qname;
|                     VAR data_msg   :string(45);
|                     VAR msg_length :integer;
|                     VAR msg_type   :sysname;
|                     VAR msg_queue  :sysname;
|                     VAR msg_que_num:integer;
|                     VAR rpy_msgq   :sysname;
|                     VAR msg_key    :num_type;
|                     VAR err_code   :error_code);nonpascal;

| (* Send message to a program message queue                       *)
| procedure QMHSNDPM (VAR msg_id     :size_type;
|                     VAR msg_file   :qname;
|                     VAR msg_data   :text_format;
|                     VAR msg_length :integer;
|                     VAR msg_type   :sysname;
|                     VAR msg_queue  :sysname;
```

```
                    VAR stack_cnt  :integer;
                    VAR msg_key    :num_type;
                    VAR err_code   :error_code);nonpascal;


(* Send message to program message queue if any error is received  *)
procedure error_check;
   begin
      if err_code.bytes_avail <> 0 then
         begin
         msg_id      := 'CPF9898';
         msg_file    := 'QCPFMSG   *LIBL';
         msg_data.msg_text := '          An error has occurred calling '
         case api_code of
            1 : msg_data.api_text := 'QUSCRTUS. ';
            2 : msg_data.api_text := 'QUSLSPL.  ';
            3 : msg_data.api_text := 'QUSPTRUS. ';
            4 : msg_data.api_text := 'QUSRSPLA. ';
            5 : msg_data.api_text := 'QUSDLTUS. ';
            6 : msg_data.api_text := 'QMHSNDM.  ';
         end;
         msg_length := 40;
         msg_type   := '*ESCAPE';
         pgm_queue  := '*';
         stack_cnt  := 1 ;
         QMHSNDPM (msg_id,
                   msg_file,
                   msg_data,
                   msg_length,
                   msg_type,
                   pgm_queue,
                   stack_cnt,
                   msg_key,
                   err_code)
      end
      else
      return;
   end;

begin
(* get parameters passed into this program using PARM function     *)
sysname_ptr :=PARM(1);
user_name   :=sysname_ptr@; (* user name to delete spooled files   *)
qname_ptr   :=PARM(2);
outq_name   :=qname_ptr@;   (* out queue to delete spooled files   *)
date_ptr    :=PARM(3);
dlt_date    :=date_ptr@;   (* delete any files older than this date*)

(* assign values to specific variables                             *)
us_name    := 'DLTOLDSPLFQTEMP     ';
us_ext_attr:= '          ';
us_init_val:= ' ';
us_auth    := '*CHANGE   ';
us_text    := '                                                ';
lspl_fmt   := 'SPLF0100';
form_type  := '*ALL      ';
user_data  := '*ALL      ';
rspla_fmt  := 'SPLA0100';

(* call QUSCRTUS to create the user space                          *)
us_size_i := 1024;
err_code.bytes_prov := 16;
api_code := 1;
replace    := '*NO       ';

QUSCRTUS(us_name,
         us_ext_attr,
         us_size_i,
         us_init_val,
         us_auth,
         us_text,
         replace,
         err_code);

error_check;



(* call QUSLSPL to list the spooled files in the user space        *)
api_code := 2;
QUSLSPL(us_name,
        lspl_fmt,
        user_name,
        outq_name,
        form_type,
        user_data,
        err_code);
error_check;

(* call QUSPTRUS to get a pointer to the user space header section *)
api_code := 3;
QUSPTRUS(us_name,us_header_ptr);

(* get info from header section of user space                      *)
list_offset     := us_header_ptr@(.124.);
num_of_entries  := us_header_ptr@(.132.);
size_of_entries := us_header_ptr@(.136.);

(* set pointer to beginning of the list entry section              *)
list_entry_ptr  := ptrto(us_header_ptr@(.list_offset.));

(* set up some variables for the delete spooled file loop          *)
rcv_var_len     := 724;       (* size of return variable           *)
qjob_name       := '*INT                    ';(* use internal id *)
splf_name       := '*INT      ';              (* use internal id *)
splf_number     := -1;        (* this will be ignored              *)
dlt_count_i     := 0;         (* initialize delete count to zero   *)

(* loop through list entry section and retrieve spooled file       *)
(* attributes, check the date, and delete if too old               *)
for count := 1 to num_of_entries do    (* check all list entries   *)
  begin
  int_job_id := list_entry_ptr@(.50.); (* get internal job id      *)
  int_spl_id := list_entry_ptr@(.66.); (* get internal splf id     *)
(* retrieve the detailed spooled file information                  *)
  api_code := 4;
  QUSRSPLA(rcv_var,
           rcv_var_len,
           rspla_fmt,
           qjob_name,
           int_job_id,
           int_spl_id,
           splf_name,
           splf_number,
           err_code);
  error_check;

  rtv_date := substr(str(rcv_var),203,7);     (* get date of file  *)

(* delete the spooled file if too old                              *)
  if rtv_date <= dlt_date then        (* check date                *)
    begin
    dlt_count_i:= dlt_count_i + 1;    (* increment delete count    *)
    d_splf     := substr(str(rcv_var),67,10); (* get spooled file  *)
                                              (* name              *)
    d_user_name:= substr(str(rcv_var),51,10); (* get user name     *)
    d_job_num  := substr(str(rcv_var),61,6);  (* get job number    *)
    d_job_name := substr(str(rcv_var),41,10); (* get job name      *)
    i_ptr      := ptrto(rcv_var(.77.));       (* get integer       *)
    d_splf_num := itoz(i_ptr@,6,0);           (* convert to zoned  *)
    d_form_type:= substr(str(rcv_var),81,10); (* get form type     *)
    d_user_data:= substr(str(rcv_var),91,10); (* get user data     *)
(* delete the spooled file                                         *)
    CLDLT(d_splf,
          d_job_num,
          d_user_name,
          d_job_name,
          d_splf_num,
          d_form_type,
          d_user_data);
  end(*if*);

(* set list entry pointer to the next entry in the list            *)
  list_entry_ptr  := ptrto(list_entry_ptr@(.82.));
end(*for*);

(* delete the user space created for the spooled file list         *)
api_code := 5;
QUSDLTUS(us_name,
         err_code);

(* Send final message to current user's message queue indicating   *)
(* number of spooled files deleted                                 *)
api_code := 6;
msg_id      := '';
msg_file    := '';
```

```
writestr(data_msg,'Number of spooled files deleted: ', dlt_count_i:10);
msg_length := 50;
msg_type   := '*INFO';
msg_queue  := '*REQUESTER';
msg_que_num:= 1;
QMHSNDM(msg_id,
        msg_file,
        data_msg,
        msg_length,
        msg_type,
        msg_queue,
        msg_que_num,
        rpy_msgq,
        msg_key,
        err_code );


end.
```

To create a Pascal program, specify the following:

```
CRTPASPGM PGM(QGPL/DLTOLDSPLF) SRCFILE(QGPL/QPASSRC)
```

System C/400 PRPQ DLTOLDSPLF Program: To delete spooled files, you can use this System C/400 PRPQ DLTOLDSPLF program:

```
/*****************************************************************/
/*  PROGRAM:  DLTOLDSPLF                                         */
/*                                                               */
/*  LANGUAGE: SYSTEM C/400                                       */
/*                                                               */
/*  DESCRIPTION:  THIS IS AN EXAMPLE PROGRAM FOR THE USE OF      */
/*                USER SPACES WRITTEN IN SYSTEM C/400.  THE FLOW */
/*                OF THIS PROGRAM IS AS FOLLOWS:                 */
/*               (1) CREATE A USER SPACE USING QUSCRTUS          */
/*               (2) GET LIST OF SPOOLED FILES IN THE USER SPACE */
/*                   USING QUSLSPL                               */
/*               (3) KEEP POINTER TO ENTRY LIST IN THE USER SPACE */
/*               (4) ENTER LOOP                                  */
/*                    RETRIEVE LIST ENTRY                        */
/*                    RETRIEVE MORE INFORMATION USING QUSRSPLA   */
/*                    IF SPOOLED FILE IS TOO OLD                 */
/*                       DELETE SPOOLED FILE                     */
/*                       INCREMENT DELETE COUNTER                */
/*                   END LOOP                                    */
/*               (5) DELETE USER SPACE                           */
/*                                                               */
/*  APIs USED:  QUSCRTUS, QUSLSPL, QUSRSPLA, QUSPTRUS, QUSDLTUS, */
/*              QMHSNDPM, AND QMHSNDM.                           */
/*                                                               */
/*****************************************************************/
#include <string.h>     /*strcpy, strncpy, strcmp         */
#include <micomput.h>   /* _DPA_Template_T, cvtnci, cvtcni */
#include <stdio.h>
#include "OPUSAPI.h"    /*Linkage info, structures for US  */
#include "opmhapi.h"    /*Linkage info, structures for MH  */


#pragma linkage(CLDLT,OS)
  void CLDLT (char file_name[10],
              char job_number[6],
              char usr_name[10],
              char job_name[10],
              char file_number[6],
              char form_type[10],
              char usr_data[10]);

  void error_check (void);

header_struct *space;
char *list_section;
splf0100 *entry_list;
spla0100 *Rcv_Spl_Var;
_DPA_Template_T d_tmp_t,k_tmp_t;
/*****************************************************************/
/* PARMS FOR CLDLT                                               */
/*****************************************************************/
char job_nmbr[6“;
char usr_nm[10“;
char job_nm[10“;
char sp_job_name[10“;
char sp_job_number[6“;
char File_Number[“ = "*LAST ";
/*****************************************************************/
/* PARMS FOR QUSLSPL                                             */
/*****************************************************************/
char frmt[8];
char usr[10];
char OutQ_Nm[20];
char ls_frm_typ[10];
char Usr_dat[10];
/*****************************************************************/
/* PARMS FOR QUSRSPLA                                            */
/*****************************************************************/
char Rcv_Var[724];
int  Rcv_lgth = 724;
char Rtv_Fmt[8];
char Qal_Jb_Nam[] = "*INT                    ";
char Splf_Name[] = "*INT      ";
int  Splf_Number = -1;
/*****************************************************************/
/* PARMS FOR QUSCRTUS                                            */
/*****************************************************************/
char spc_name[20];
char ext_atr[10];
int  initial_size;
char initial_value[1];
char auth[10];
char desc[50];
char replace[10];
/*****************************************************************/
/* PARMS FOR QMHSNDPM AND QMHSNDM                                */
/*****************************************************************/
char msg_id[7];
char msg_fl_name[20];
char msg_data[50];
int  msg_data_len;
char msg_type[10];
char pgm_queue[10];
int  pgm_stk_cnt;
char msg_key[4];
/*****************************************************************/
/* PARMS FOR QMHSNDM                                             */
/*****************************************************************/
int  msg_q_num;
char msg_queue[20];
char rpy_mq[10];
/*****************************************************************/
/* MISCELLANEOUS VARIABLES                                       */
/*****************************************************************/
char pack_dlt_count[15];
int  dlt_cnt;
int count;
char dlt_date[7];
char spc_date[7];
int  api_code;
typedef struct {
       int bytes_prov;
       int bytes_avail;
       char except_id[7];
       char reserved[1];
       char except_data[50];
       } error_code;

error_code err_code;

/*****************************************************************/
/* PROCEDURE TO COPY DATA FROM ONE STRING TO ANOTHER             */
/*****************************************************************/
void strncpyn(char string1[],char string2¢¢,int strpos,int cpylngth)
{
 int x = 0;
 while (x < cpylngth)
  string1[x++]=string2¢strpos++¢;
}

/*****************************************************************/
/* PROCEDURE TO CHECK THE ERRCODE RETURNED FROM CALLS TO APIs    */
/*****************************************************************/
void error_check(void)
{
if (err_code.bytes_avail != 0){
     strncpy(msg_id,"CPF9898",7);
     strncpy(msg_fl_name,"QCPFMSG   *LIBL     ",20);
```

```
      strncpy(msg_data,"An error has occurred calling ",29);
      switch (api_code){
         case  1  : strncat(msg_data,"QUSCRTUS.",9);
         case  2  : strncat(msg_data,"QUSLSPL. ",9);
         case  3  : strncat(msg_data,"QUSPTRUS.",9);
         case  4  : strncat(msg_data,"QUSRSPLA.",9);
         case  5  : strncat(msg_data,"QUSDLTUS.",9);
         case  6  : strncat(msg_data,"QMHSNDM. ",9);
         default  : strncat(msg_data,"UNKNOWN. ",9);
         }
      msg_data_len = 38;
      strncpy(msg_type,"*ESCAPE   ",10);
      strncpy(pgm_queue,"*         ",10);
      pgm_stk_cnt = 1;

      QMHSNDPM(msg_id,msg_fl_name,msg_data,msg_data_len,msg_type,
               pgm_queue,pgm_stk_cnt,msg_key,&err_code);
      }
}

/*********************************************************************/
/* START OF MAINLINE                                                 */
/*********************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
/*********************************************************************/
/*  Assign the parameters for the conversion routines                */
/*********************************************************************/
d_tmp_t.Type     = T_Packed;
d_tmp_t.Length   = 13;
d_tmp_t.reserved = 0;
k_tmp_t.Type     = T_Zoned;
k_tmp_t.Length   = 6;
k_tmp_t.reserved = 0;

/*********************************************************************/
/*  Read in and assign the command-line arguments to respective      */
/*  variables                                                        */
/*********************************************************************/
strncpy(usr,argv[1],10);
strncpy(OutQ_Nm,argv[2],20);
strncpy(dlt_date,argv[3],7);

/*********************************************************************/
/*  Assign value to specific variables in the program                */
/*********************************************************************/
strcpy(spc_name,"DLTOLDSPLFQTEMP     ");
memset(ext_atr,' ',10);
initial_size = 1024;
strcpy(initial_value," ");
strcpy(auth,"*CHANGE   ");
memset(desc,' ',50);
strcpy(frmt,"SPLF0100");
strcpy(replace,"*YES      ");
strcpy(ls_frm_typ,"*ALL      ");
strcpy(Usr_dat,"*ALL      ");
strcpy(Rtv_Fmt,"SPLA0100");

/*********************************************************************/
/*  Call external program to create a user space                     */
/*********************************************************************/
err_code.bytes_prov = 16;
api_code = 1;
QUSCRTUS(spc_name,ext_atr,initial_size,initial_value,auth,desc,replace,
         &err_code);
/*********************************************************************/
/*  Call external program to list spooled files into user space      */
/*********************************************************************/
api_code = 2;
QUSLSPL(spc_name,frmt,usr,OutQ_Nm,ls_frm_typ,Usr_dat,&err_code);
/*********************************************************************/
/*  Call external program to get a pointer to the user space         */
/*  and get addressability to the list data section.                 */
/*********************************************************************/
api_code = 3;
QUSPTRUS(spc_name,&space,&err_code);
list_section = (char *)space;
list_section = list_section + space->list_section_offset;
entry_list = (splf0100 *) list_section;
dlt_cnt = 0;
count = 1;
/*********************************************************************/
/*  Loop through the entry list and delete old spooled files         */
/*********************************************************************/
while (count <= space->number_of_entries) {
/*********************************************************************/
/*  Call external program to retrieve more spool information         */
/*********************************************************************/
 api_code = 4;
 QUSRSPLA(Rcv_Var,Rcv_lgth,Rtv_Fmt,Qal_Jb_Nam,
          entry_list->int_job_ID,entry_list->int_splf_ID,
          Splf_Name,Splf_Number,&err_code);
 Rcv_Spl_Var = (spla0100 *)Rcv_Var;
 strncpy(spc_date,Rcv_Spl_Var->date_file_open,7);
/*********************************************************************/
/*  If spooled file is too old delete it                             */
/*********************************************************************/
 if (strncmp(spc_date,dlt_date,7) <= 0 ) {
   strncpy(job_nm,Rcv_Spl_Var->job_name,10);
   strncpy(job_nmbr,Rcv_Spl_Var->job_number,6);
   strncpy(usr_nm,Rcv_Spl_Var->usr_name,10);
   strncpy(sp_job_name,Rcv_Spl_Var->splf_name,10);
   cvtnci(sp_job_number,Rcv_Spl_Var->splf_number,k_tmp_t);
   CLDLT(sp_job_name,job_nmbr,usr_nm,
         job_nm,sp_job_number,ls_frm_typ,Usr_dat);
   dlt_cnt++;
  } /*IF*/
 strcpy(spc_date," ");
 count++;
 entry_list++;
} /*WHILE*/
/*********************************************************************/
/*  Remove the user space                                            */
/*********************************************************************/
api_code = 5;
QUSDLTUS(spc_name, &err_code);

/*********************************************************************/
/*  Send final message to user indicating number of spooled files    */
/*  deleted.                                                         */
/*********************************************************************/
api_code = 6;
strncpy(msg_id,"       ",7);
strncpy(msg_fl_name,"                    ",20);
sprintf(msg_data,"Number of spooled files deleted:  %d", dlt_cnt);
msg_data_len = strlen(msg_data);
strncpy(msg_type,"*INFO     ",10);
strncpy(msg_queue,"*REQUESTER          ",20);
msg_q_num = 1;
strncpy(rpy_mq,"          ",10);

QMHSNDM(msg_id,msg_fl_name,msg_data,msg_data_len,msg_type,
        msg_queue,msg_q_num,rpy_mq,msg_key, &err_code);
```

To create a System C/400 PRPQ program, specify the following:

```
CRTSYSCTXT TXT(QGPL/DLTOLDSPLF) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/DLTOLDSPLF) TXT(QGPL/DLTOLDSPLF)
```

CL Delete (CLDLT) Program

The DLTOLDSPLF program, written in RPG, COBOL, Pascal, or System C/400 PRPQ, calls a CL program named CLDLT. The CLDLT program deletes the spooled files and the user space. The following is the CL source for the CLDLT program.

```
/*********************************************************************/
/*                                                                   */
/*  PROGRAM:  CLDLT                                                  */
/*                                                                   */
/*  LANGUAGE:  CL                                                    */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM WILL DELETE A SPECIFIC SPOOLED FILE   */
/*                USING THE DLTSPLF COMMAND AND SEND A MESSAGE WHEN  */
/*                THE FILE IS DELETED.                               */
/*                                                                   */
```

```
| /*                                                                  */
| /*******************************************************************/
| /*                                                                  */
| PGM (&FILNAM &JOBNUM &USRNAM &JOBNAM &FILNUM &FRMTYP &USRDTA)
| /*                                                                  */
| /* ************************************************************** */
| /*                                                                  */
| /*  DECLARE SECTION                                                 */
| /*                                                                  */
| /*******************************************************************/
| /*                                                                  */
|  DCL &FILNAM *CHAR 10
|  DCL &JOBNUM *CHAR 6
|  DCL &USRNAM *CHAR 10
|  DCL &JOBNAM *CHAR 10
|  DCL &FILNUM *CHAR 6
|  DCL &FRMTYP *CHAR 10
|  DCL &USRDTA *CHAR 10
|  MONMSG CPF0000
| /*                                                                  */
| /*******************************************************************/
| /*                                                                  */
| /* EXECUTABLE CODE                                                 */
| /*                                                                  */
| /*******************************************************************/
| /*                                                                  */

|          DLTSPLF    FILE(&FILNAM)                                  +
|                     JOB(&JOBNUM/&USRNAM/&JOBNAM)                   +
|                     SPLNBR(&FILNUM)                                +
|                     SELECT(&USRNAM *ALL &FRMTYP &USRDTA)
|          SNDPGMMSG  MSG('Spool file ' *CAT &JOBNUM *CAT '/'       +
|                          *CAT &USRNAM *CAT '/' *CAT &JOBNUM *CAT +
|                          ' deleted.')                              +
|                     TOUSR(*REQUESTER)
| ENDPGM
```

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CLDLT) SRCFILE(QGPL/QCLSRC)
```

Changing an Active Job

This command interface to the Change Active Jobs (CHGACTJOB) program can reduce the run priority of active jobs with the same name. You can also reduce the run priority of jobs using a specified user name. You may:

- Specify a job name or the *ALL value.
- Specify the user name as the *ALL value.
- Use the default run priority of 99.

The CHGACTJOB command ensures that one of the following is true:

- Not all jobs were specified.
- The *ALL value was not specified for the JOB parameter.
- The *ALL value was not specified for the USER parameter.

This example uses the following APIs:

- Create User Space (QUSCRTUS)
- List Job (QUSLJOB)
- Retrieve User Space (QUSRTVUS)
- Retrieve Job Information (QUSRJOBI)

The following is the message description needed for the Change Active Jobs (CHGACTJOB) command:

```
ADDMSGD MSGID(USR3C01) MSGF(QCPFMSG) +
MSG('JOB(*ALL) is not valid with USER(*ALL)') SEV(30)
```

The following is the command definition for the CHGACTJOB command:

```
CMD         PROMPT('Change Active Jobs')
            /* CPP CHGACTJOB */
PARM        KWD(JOB) TYPE(*NAME) LEN(10) +
              SPCVAL((*ALL)) MIN(1) +
              PROMPT('Job name:')
PARM        KWD(USER) TYPE(*NAME) LEN(10) DFT(*ALL) +
              SPCVAL((*ALL) (*CURRENT)) PROMPT('User +
              name:')
PARM        KWD(RUNPTY) TYPE(*DEC) LEN(5 0) DFT(99) +
              RANGE(00 99) PROMPT('Run priority:')
DEP         CTL(&USER *EQ *ALL) PARM((&JOB *NE *ALL)) +
              NBRTRUE(*EQ 1) MSGID(USR3C01)
```

To create the command, specify the following:

```
CRTCMD CMD(QGPL/CHGACTJOB) PGM(QGPL/CHGACTJOB) +
  SRCFILE(QGPL/CMDSRC)
```

The following is the command-processing program that is written in CL to list the active jobs and reduce the run priority if necessary:

```
/* ************************************************************** */
/* PROGRAM:  CHGACTJOB                                            */
/*                                                                */
/* LANGUAGE:  CL                                                  */
/*                                                                */
/* DESCRIPTION:  THIS PROGRAM WILL REDUCE THE RUN PRIORITY OF ACTIVE */
/*               JOBS WITH THE SAME NAME.                         */
/*                                                                */
/* APIs USED:  QUSCRTUS, QUSLJOB, QUSRTVUS, QUSRJOBI              */
/*                                                                */
/* ************************************************************** */
            PGM         PARM(&JOB &USER &RUNPTY)

/*                                                                */
/* Input parameters                                               */
/*                                                                */

            DCL         VAR(&JOB) TYPE(*CHAR) LEN(10) +
                          /* Input job name */
            DCL         VAR(&USER) TYPE(*CHAR) LEN(10) +
                          /* Input user name */
            DCL         VAR(&RUNPTY) TYPE(*DEC) LEN(5 0) +
                          /* Input run priority */

/*                                                                */
/* Local variables                                                */
/*                                                                */

            DCL         VAR(&RJOB) TYPE(*CHAR) LEN(10) +
                          /* Retrieve job name */
            DCL         VAR(&RUSER) TYPE(*CHAR) LEN(10) +
                          /* Retrieve user name */
            DCL         VAR(&RNBR) TYPE(*CHAR) LEN(6) +
                          /* Retrieve job number */
            DCL         VAR(&RUNPTYC) TYPE(*CHAR) LEN(5) +
                          /* Input run priority in character form */
            DCL         VAR(&RUNPTY8) TYPE(*DEC) LEN(8 0) +
                          /* Retrieve run priority after convert from +
                             binary 4 */
            DCL         VAR(&RUNPTY5) TYPE(*DEC) LEN(5 0) +
                          /* Retrieve run priority in decimal 5,0 +
                             form */
            DCL         VAR(&RUNPTY5C) TYPE(*CHAR) LEN(5) +
                          /* Retrieve run priority in character form */
            DCL         VAR(&RUNPTY4) TYPE(*CHAR) LEN(4) +
                          /* Retrieve run priority in binary 4 form */
            DCL         VAR(&NUMBER) TYPE(*CHAR) LEN(6) +
                          /* Current job number */
            DCL         VAR(&USRSPC) TYPE(*CHAR) LEN(20) +
                          VALUE('CHGA      QTEMP     ') +
                          /* User space name for APIs */
            DCL         VAR(&EUSRSPC) TYPE(*CHAR) LEN(10) +
                          /* User space name for commands */
            DCL         VAR(&JOBNAME) TYPE(*CHAR) LEN(26) +
```

```
                VALUE('                    *ALL  ') +
                /* Full job name for list job  */
          DCL   VAR(&BIN4) TYPE(*CHAR) LEN(4) +
                /* Number of jobs for list job and +
                   User space offset in binary 4 form */
          DCL   VAR(&LOOP) TYPE(*DEC) LEN(8 0) +
                /* Number of jobs from list job */
          DCL   VAR(&DEC8) TYPE(*DEC) LEN(8 0) +
                /* User space offset in decimal 8,0 form */
          DCL   VAR(&ELEN) TYPE(*DEC) LEN(8 0) +
                /* List job entry length in decimal 8,0 +
                   form */
          DCL   VAR(&ELENB) TYPE(*CHAR) LEN(4) +
                /* List job entry length in binary 4 +
                   form */
          DCL   VAR(&LJOBE) TYPE(*CHAR) LEN(52) +
                /* Retrieve area for list job entry */
          DCL   VAR(&INTJOB) TYPE(*CHAR) LEN(16) +
                /* Retrieve area for internal job id */
          DCL   VAR(&JOBI) TYPE(*CHAR) LEN(104) +
                /* Retrieve area for job information */
          DCL   VAR(&JOBTYPE) TYPE(*CHAR) LEN(1) +
                /* Job type */

/*                                                              */
/* Start of executable code                                     */
/*                                                              */

/*                                                              */
/* Retrieve job number to use for local user space name         */
/*                                                              */

          RTVJOBA   NBR(&NUMBER)
          CHGVAR    VAR(%SST(&USRSPC 5 6)) VALUE(&NUMBER)
          CHGVAR    VAR(&EUSRSPC) VALUE(%SST(&USRSPC 1 10))

/*                                                              */
/* Delete user space if it already exists                       */
/*                                                              */

          DLTUSRSPC  USRSPC(QTEMP/&EUSRSPC)
          MONMSG CPF0000

/*                                                              */
/* Create user space                                            */
/*                                                              */

          CALL QUSCRTUS (&USRSPC 'CHGACTJOB ' X'00000100' ' ' +
                         '*ALL      ' +
                         'CHGACTJOB TEMPORARY USER SPACE-
                ')

/*                                                              */
/* Set up job name for list jobs                                */
/*                                                              */

          CHGVAR    VAR(%SST(&JOBNAME 1 10)) VALUE(&JOB)
          CHGVAR    VAR(%SST(&JOBNAME 11 10)) VALUE(&USER)

/*                                                              */
/* List active jobs with job name specified                     */
/*                                                              */

          CALL QUSLJOB  (&USRSPC 'JOBL0100' &JOBNAME +
                         '*ACTIVE   ') .

/*                                                              */
/* Retrieve number of entries returned. Convert to decimal and  */
/* if zero go to NOJOBS label to send out 'No jobs' message.    */
/*                                                              */

          CALL QUSRTVUS (&USRSPC X'00000085' X'00000004' +
                         &BIN4)
          CHGVAR    &LOOP      %BINARY(&BIN4)
          IF        COND(&LOOP = 0) THEN(GOTO CMDLBL(NOJOBS))

/*                                                              */
/* Retrieve list entry length, convert to decimal.              */
/* Retrieve list entry offset, convert to decimal, and add one  */
/* to set the position.                                         */
/*                                                              */

          CALL QUSRTVUS (&USRSPC X'00000089' X'00000004' +
                         &ELENB)
          CHGVAR    &ELEN      %BINARY(&BIN4)
          CALL QUSRTVUS (&USRSPC X'0000007D' X'00000004' +
                         &BIN4)
          CHGVAR    &DEC8      %BINARY(&BIN4)
          CHGVAR    VAR(&DEC8) VALUE(&DEC8 + 1)

/*                                                              */
/* Loop for the number of jobs until no more jobs then go to    */
/* ALLDONE label                                                */
/*                                                              */

STARTLOOP:  IF (&LOOP = 0) THEN(GOTO ALLDONE)

/*                                                              */
/* Convert decimal position to binary 4 and retrieve list job entry */
/*                                                              */

          CHGVAR    %BINARY(&BIN4)   &DEC8
          CALL QUSRTVUS (&USRSPC &BIN4 &ELENB +
                         &LJOBE)

/*                                                              */
/* Copy internal job identifier and retrieve job information for */
/* basic performance information.                               */
/*                                                              */

          CHGVAR    VAR(&INTJOB) VALUE(%SST(&LJOBE 27 16))
          CALL QUSRJOBI (&JOBI X'00000068' 'JOBI0100' +
                         '*INT                    ' +
                         &INTJOB)

/*                                                              */
/* Copy job type and if subsystem monitor, spool reader, system job, */
/* spool writer, or SCPF system job then loop to next job       */
/*                                                              */

          CHGVAR    VAR(&JOBTYPE) VALUE(%SST(&JOBI 61 1))
          IF        COND((&JOBTYPE = 'M') *OR (&JOBTYPE = 'R')  +
                         *OR (&JOBTYPE = 'S') *OR (&JOBTYPE = 'W')  +
                         *OR (&JOBTYPE = 'X')) +
                          THEN(GOTO CMDLBL(ENDLOOP))

/*                                                              */
/* Copy run priority, convert to decimal, convert to decimal 5,0, */
/* and if request run priority is less than or equal to the current */
/* run priority then loop to next job.                          */
/*                                                              */

          CHGVAR    VAR(&RUNPTY4) VALUE(%SST(&JOBI 65 4))
          CHGVAR    &RUNPTY8      %BINARY(&RUNPTY4)
          CHGVAR    VAR(&RUNPTY5) VALUE(&RUNPTY8)
          IF        COND(&RUNPTY5 *GE &RUNPTY) THEN(GOTO +
                      CMDLBL(ENDLOOP))

/*                                                              */
/* Retrieve job name, convert to run priority to character,  change */
/* the job run priority and seen message stating the run priority */
/* was changed.                                                 */
/*                                                              */

          CHGVAR    VAR(&RJOB) VALUE(%SST(&JOBI 9 10))
          CHGVAR    VAR(&RUSER) VALUE(%SST(&JOBI 19 10))
          CHGVAR    VAR(&RNBR) VALUE(%SST(&JOBI 29 6))
          CHGVAR    VAR(&RUNPTYC) VALUE(&RUNPTY)
          CHGVAR    VAR(&RUNPTY5C) VALUE(&RUNPTY5)
          CHGJOB    JOB(&RNBR/&RUSER/&RJOB) RUNPTY(&RUNPTYC)
          MONMSG    MSGID(CPF1343) EXEC(GOTO CMDLBL(ENDLOOP))
          SNDPGMMSG MSG('Job' *BCAT &RNBR *TCAT '/' *TCAT +
                      &RUSER *TCAT '/' *TCAT &RJOB *BCAT 'run +
                      priority was change from' *BCAT &RUNPTY5C +
                      *BCAT 'to' *BCAT &RUNPTYC *TCAT '.')

/*                                                              */
/* At end of loop set new decimal position to next entry and    */
/* decrement loop counter by one.                               */
/*                                                              */

ENDLOOP:  CHGVAR    VAR(&DEC8) VALUE(&DEC8 + &ELEN)
          CHGVAR    VAR(&LOOP) VALUE(&LOOP - 1)
          GOTO      CMDLBL(STARTLOOP)

/*                                                              */
```

```
/* Send message that no jobs were found.                          */
/*                                                                */

NOJOBS:     SNDPGMMSG  MSG('No jobs found.')

/*                                                                */
/* All done. Now delete temporary user space that we created.     */
/*                                                                */

ALLDONE:    DLTUSRSPC  USRSPC(QTEMP/&EUSRSPC)
            MONMSG CPF0000
            ENDPGM
```

The program can be changed to change the run priority by removing the IF statement to compare the current and requested run priority.

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CHGACTJOB) SRCFILE(QGPL/QCLSRC)
```

You can change the command to:

- Specify a different printer device.
- Specify a different output queue.
- Specify different job attributes that the Change Job (CHGJOB) command can change.
- List only jobs on an output queue and remove the spooled files.
- Provide a menu to select jobs to be changed.

Changing a Job Schedule Entry

This command interface to the Change Job Schedule Entry User (CHGSCDEUSR) program can change the USER parameter in the job schedule entry. You may:

- Specify a job schedule entry name
- Specify a generic job schedule entry name
- Specify the *ALL value

This example uses the following APIs:

- Create User Space (QUSCRTUS)
- List Job Schedule Entries (QWCLSCDE)
- Retrieve User Space (QUSRTVUS)

The following is the command definition for the CHGSCDEUSR command:

```
CMD         PROMPT('Change Job Schedule Entry User')
            /* CPP CHGSCDEUSR */
PARM        KWD(JOB) TYPE(*GENERIC) LEN(10) +
              SPCVAL((*ALL)) +
              MIN(1) PROMPT('Job name:')
PARM        KWD(OLDUSER) TYPE(*NAME) LEN(10) +
              MIN(1) PROMPT('Old user name:')
PARM        KWD(NEWUSER) TYPE(*NAME) LEN(10) +
              MIN(1) PROMPT('New user name:')
```

To create the command, specify the following:

```
CRTCMD CMD(QGPL/CHGSCDEUSR) PGM(QGPL/CHGSCDEUSR) +
  SRCFILE(QGPL/QCMDSRC)
```

The following is the command-processing program that is written in CL to list the job schedule entries and change the user if necessary:

```
/* ************************************************************ */
/* PROGRAM:  CHGSCDEUSR                                         */
/*                                                              */
/* LANGUAGE:  CL                                                */
/*                                                              */
/* DESCRIPTION:  THIS PROGRAM WILL CHANGE THE USER FOR A LIST OF */
/*               JOB SCHEDULE ENTRIES.                          */
/*                                                              */
/* APIs USED:  QUSCRTUS, QWCLSCDE, QUSRTVUS                     */
/*                                                              */
/* ************************************************************ */
            PGM        PARM(&JOBNAME &OLDUSER &NEWUSER)

/*                                                              */
/* Input parameters are as follows:                             */
/*                                                              */

            DCL        VAR(&JOBNAME) TYPE(*CHAR) LEN(10) /* Input +
                         job name */
            DCL        VAR(&OLDUSER) TYPE(*CHAR) LEN(10) /* Input +
                         old user name */
            DCL        VAR(&NEWUSER) TYPE(*CHAR) LEN(10) /* Input +
                         new user name */

/*                                                              */
/* Local variables are as follows:                              */
/*                                                              */

            DCL        VAR(&USRSPC) TYPE(*CHAR) LEN(20) +
                         VALUE('CHGSCDEUSRQTEMP     ') /* User +
                         space name for APIs */
            DCL        VAR(&CNTHDL) TYPE(*CHAR) LEN(16) +
                         VALUE('                ') /* Continuation +
                         handle */
            DCL        VAR(&NUMENTB) TYPE(*CHAR) LEN(4) /* Number +
                         of entries from list job schedule entries +
                         in binary form */
            DCL        VAR(&NUMENT) TYPE(*DEC) LEN(8 0) /* Number +
                         of entries from list job schedule entries +
                         in decimal form */
            DCL        VAR(&HDROFFB) TYPE(*CHAR) LEN(4) /* Offset +
                         to the header portion of the user space in +
                         binary form */
            DCL        VAR(&HDRLENB) TYPE(*CHAR) LEN(4) /* Length +
                         to the header portion of the user space in +
                         binary form */
            DCL        VAR(&GENHDR) TYPE(*CHAR) LEN(140) /* Generic +
                         header information from the user space */
            DCL        VAR(&HDRINFO) TYPE(*CHAR) LEN(26) /* Header +
                         information from the user space */
            DCL        VAR(&LSTSTS) TYPE(*CHAR) LEN(1) /* Status +
                         of the list in the user space */
            DCL        VAR(&OFFSETB) TYPE(*CHAR) LEN(4) /* Offset +
                         to the list portion of the user space in +
                         binary form */
            DCL        VAR(&STRPOSB) TYPE(*CHAR) LEN(4) /* Starting +
                         position in the user space  in binary form */
            DCL        VAR(&ELENB) TYPE(*CHAR) LEN(4) /* List job +
                         entry length in binary 4 form */
            DCL        VAR(&LENTRY) TYPE(*CHAR) LEN(1156) /* +
                         Retrieve area for list job schedule entry */
            DCL        VAR(&INFOSTS) TYPE(*CHAR) LEN(1) /* Retrieve +
                         area for information status */
            DCL        VAR(&JOBNAM) TYPE(*CHAR) LEN(10) /* Retrieve +
                         area for job name */
            DCL        VAR(&ENTRY#) TYPE(*CHAR) LEN(6) /* Retrieve +
                         area for entry number  */
            DCL        VAR(&USERNM) TYPE(*CHAR) LEN(10) /* Retrieve +
                         area for user name */

/*                                                              */
/* Start of code                                                */
/*                                                              */

/*                                                              */
/* You may want to monitor for additional messages here.        */
/*                                                              */

/*                                                              */
/* This creates the user space.  The user space will be 256 bytes */
/* and will be initialized to blanks.                           */
/*                                                              */
```

```
            CALL       PGM(QUSCRTUS) PARM(&USRSPC 'CHGSCDEUSR' +
                         X'00000100' ' ' '*ALL      ' 'CHGSCDEUSR +
                         TEMPORARY USER SPACE                    ')
            MONMSG     MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))

/*                                                                */
/* This lists job schedule entries of the name specified.         */
/*                                                                */

PARTLIST:   CALL       PGM(QWCLSCDE) PARM(&USRSPC 'SCDL0200' +
                         &JOBNAME &CNTHDL 0)

/*                                                                */
/* Retrieve the generic header from the user space.               */
/*                                                                */

            CALL       PGM(QUSRTVUS) PARM(&USRSPC X'00000001' +
                         X'0000008C' &GENHDR)
            MONMSG     MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))
/*                                                                */
/* Get the information status for the list from the generic header. */
/* If it is incomplete, go to BADLIST label and send out 'Bad list' */
/* message.                                                       */
/*                                                                */

            CHGVAR     VAR(&LSTSTS) VALUE(%SST(&GENHDR 104 1))
            IF         COND(&LSTSTS = 'I') THEN(GOTO CMDLBL(BADLIST))

/*                                                                */
/* Get the number of entries returned. Convert to decimal and     */
/* if zero go to NOENTS label to send out 'No entries' message.   */
/*                                                                */

            CHGVAR     VAR(&NUMENTB) VALUE(%SST(&GENHDR 133 4))
            CHGVAR     VAR(&NUMENT) VALUE(%BIN(&NUMENTB))
            IF         COND(&NUMENT = 0) THEN(GOTO CMDLBL(NOENTS))

/*                                                                */
/* Get the list entry length and the list entry offset.           */
/* These values are used to set up the starting position.         */
/*                                                                */

            CHGVAR     VAR(&ELENB) VALUE(%SST(&GENHDR 137 4))
            CHGVAR     VAR(&OFFSETB) VALUE(%SST(&GENHDR 125 4))
            CHGVAR     VAR(%BIN(&STRPOSB)) VALUE(%BIN(&OFFSETB) + 1)

/*                                                                */
/* This loops for the number of entries until no more entries are */
/* found and goes to the ALLDONE label.                           */
/*                                                                */

STARTLOOP:  IF         COND(&NUMENT = 0) THEN(GOTO CMDLBL(PARTCHK))

/*                                                                */
/* This retrieves the list entry.                                 */
/*                                                                */
            CALL       PGM(QUSRTVUS) PARM(&USRSPC &STRPOSB &ELENB +
                         &LENTRY)
            MONMSG     MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))
/*                                                                */
/* This copies the information status, job name, entry number, and */
/* user name.                                                     */
/*                                                                */

            CHGVAR     VAR(&INFOSTS) VALUE(%SST(&LENTRY 1 1))
            CHGVAR     VAR(&JOBNAM)  VALUE(%SST(&LENTRY 2 10))
            CHGVAR     VAR(&ENTRY#)  VALUE(%SST(&LENTRY 12 10))
            CHGVAR     VAR(&USERNM)  VALUE(%SST(&LENTRY 547 10))

/*                                                                */
/* This checks to make sure the list entry contains the user name. */
/* If it does, the user name is compared to the old user name     */
/* passed in.  If either of these checks fails, this entry will   */
/* be skipped.                                                    */
/*                                                                */

            IF         COND(&INFOSTS ¬= ' ') THEN(GOTO +
                         CMDLBL(ENDLOOP))

            IF         COND(&USERNM ¬= &OLDUSER) THEN(GOTO +
                         CMDLBL(ENDLOOP))

/*                                                                */
/* This code will issue the CHGJOBSCDE command for the entry.     */
/*                                                                */

            CHGJOBSCDE JOB(&JOBNAM) ENTRYNBR(&ENTRY#) USER(&NEWUSER)
            MONMSG     MSGID(CPF1620) EXEC(GOTO CMDLBL(NOCHG))
            SNDPGMMSG  MSG('Entry' *BCAT &JOBNAM *BCAT &ENTRY# +
                         *BCAT 'was changed.')
            GOTO       CMDLBL(ENDLOOP)
NOCHG:      SNDPGMMSG  MSG('Entry' *BCAT &JOBNAM *BCAT &ENTRY# +
                         *BCAT 'was NOT changed.')

/*                                                                */
/* At end of loop, set new decimal position to the next entry and */
/* decrement the loop counter by one.                             */
/*                                                                */

ENDLOOP:    CHGVAR     VAR(%BIN(&STRPOSB)) VALUE(%BIN(&STRPOSB) +
                         + %BIN(&ELENB))
            CHGVAR     VAR(&NUMENT) VALUE(&NUMENT - 1)
            GOTO       CMDLBL(STARTLOOP)

/*                                                                */
/* This sends a message that no entries were found.               */
/*                                                                */

NOENTS:     SNDPGMMSG  MSG('No entries found.')
            GOTO       CMDLBL(ALLDONE)

/*                                                                */
/* This sends a message that the list was incomplete.             */
/*                                                                */

BADLIST:    SNDPGMMSG  MSG('Incomplete list in the user space. +
                            See joblog for details.')
            GOTO       CMDLBL(ALLDONE)

/*                                                                */
/* This sends a message that an unexpected error occurred.        */
/*                                                                */

ERROR:      SNDPGMMSG  MSG('Unexpected error. +
                            See joblog for details.')
            GOTO       CMDLBL(ALLDONE)

/*                                                                */
/* This will check for a partial list in the user space and       */
/* finish processing the rest of the list.                        */
/*                                                                */

PARTCHK:    IF         COND(&LSTSTS = 'C') THEN(GOTO CMDLBL(ALLDONE))
/*                                                                */
/* Retrieve the header information from the user space.           */
/* Use this information to get the rest of the list.              */
/*                                                                */

            CHGVAR     VAR(&HDROFFB) VALUE(%SST(&GENHDR 121 4))
            CHGVAR     VAR(&HDRLENB) VALUE(%SST(&GENHDR 117 4))
            CALL       PGM(QUSRTVUS) PARM(&USRSPC &HDROFFB +
                         &HDRLENB &HDRINFO)
            MONMSG     MSGID(CPF3C00) EXEC(GOTO CMDLBL(ERROR))
            CHGVAR     VAR(&CNTHDL) VALUE(%SST(&HDRINFO 11 16))
            GOTO       CMDLBL(PARTLIST)


/*                                                                */
/* All done.  Now the temporary user space is deleted.            */
/*                                                                */

ALLDONE:    DLTUSRSPC  USRSPC(QTEMP/%SST(&USRSPC 1 10))
            MONMSG     MSGID(CPF0000)
            ENDPGM
```

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CHGSCDEUSR) SRCFILE(QGPL/QCLSRC)
```

You can change the command to:

- Specify different parameters that the Change Job Schedule Entry (CHGJOBSCDE) command can change.
- Provide a menu to select job schedule entries to be changed.

Creating Your Own Telephone Directory

To create a telephone directory, you must use the following $USIDXCRT System C/400 PRPQ program to create a user index, and you must use the $USIDXEX System C/400 PRPQ program to insert the entries into a telephone directory.

To set up the program to create a user index, specify the following:

```
/*********************************************************************/
/*  PROGRAM:  $USIDXCRT                                              */
/*                                                                   */
/*  LANGUAGE:  System C/400 PRPQ                                     */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM CREATES A USER INDEX NAMED TESTIDX    */
/*   IN THE LIBRARY QGPL.                                            */
/*                                                                   */
/*  APIs USED:  QUSCRTUI                                             */
/*                                                                   */
/*********************************************************************/
#include "OPENAPI.H"   /* Linkage info, structures     */
main()
{
/*********************************************************************/
/*  This sets up the parameters to call the QUSCRTUI API.            */
/*********************************************************************/
char idx_name[]= "TESTIDX   QGPL      ";
char ext_atr[]= "TESTER    ";
char entry_lgth_att[] = "F";
int entry_lngth = 50;
char key_insert[] = "1";
int key_lngth = 15;
char imm_update[] = "0";
char optim[] = "0";
char auth[] = "*CHANGE   ";
char desc[] = "Description .....    ";
/*********************************************************************/
/*  This calls the QUSCRTUI API to create the user index.            */
/*********************************************************************/
  QUSCRTUI(idx_name,ext_atr,entry_lgth_att,entry_lngth,key_insert,
           key_lngth,imm_update,optim,auth,desc);
}
```

To compile the program that creates the user index, specify the following:

```
CRTSYSCTXT TXT(QGPL/$USIDXCRT) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/$USIDXCRT) FROMTXT(QGPL/$USIDXCR)
```

To insert entries into the user index, use the following System C/400 PRPQ program:

```
/*********************************************************************/
/*  PROGRAM:  $USIDXEX                                               */
/*                                                                   */
/*  LANGUAGE:  System C/400 PRPQ                                     */
/*                                                                   */
/*  DESCRIPTION:  This program uses a user index to keep track of    */
/*    names and telephone numbers.  Two operations are shown in      */
/*    this example.  The first operation inserts an entry into       */
/*    the user index, and the second operation finds the index       */
/*    entry.                                                         */
/*    The user index is keyed on the last name, so you can enter as  */
/*    many of the names as you know, and the program lists all the   */
/*    entries matching your string in alphabetic order.              */
/*                                                                   */
/*  APIs USED:  None                                                 */
/*                                                                   */
/*********************************************************************/
#define C400
#include "C400.h" /* printf,gets,clibinit,clibterm,_Usridx,_AUTH_ALL*/
#include <string.h>   /*strlen, strcpy, strncmp          */
#include <miindex.h>  /*_IIX_Opt_List_T,insinxen,fndinxen*/
#include <miptrnam.h> /*_SYSPTR, rslvsp                  */
#include <stdlib.h>   /*malloc   */
#include <ctype.h>    /*toupper  */
_SYSPTR index;
_IIX_Opt_List_T  ins_option_list;
_IIX_Opt_List_T  *fnd_option_list;
char Name_And_Num[50];
char In_Name[50];
char Out_Num[5000];
char response[1];
char name[35];
char number[15];
int  Ent_Found,count,start,length_of_entry;


/*********************************************************************/
/* This copies the cpylngth elements of string2 into the             */
/* new string, string1, starting from position strpos.               */
/*********************************************************************/
void strncpyn(char string[],char string2[,
 int strpos, int cpylngth)
{
 int x = 0;
 while (x < cpylngth)
  string1[x++]=string2[strpos++];
} /*strncpyn*/


/*********************************************************************/
/* This converts any string into uppercase, where it is applicable. */
/*********************************************************************/
void convert_case(char string1[])
{
 int x = 0;
 while (x < (strlen(string1))) {
  string1[x] = toupper(string1[x]);
  x++;
 } /*while*/
} /*convert_case*/


main()
{
fnd_option_list =
malloc(sizeof(_IIX_Opt_List_T)+99*sizeof(_IIX_Entry_T));
/*********************************************************************/
/*  This resolves to the index created in $USIDXCRT.                 */
/*********************************************************************/
  index = rslvsp(_Usridx,"TESTIDX","QGPL",_AUTH_ALL);
/*********************************************************************/
/*  This sets up the insert option list.                             */
/*********************************************************************/
  ins_option_list.Rule = _Insert_Repl;
  ins_option_list.Arg_Length = 50;
  ins_option_list.Occ_Count = 1;
  ins_option_list.Entry[0].Entry_Length = 50;
  ins_option_list.Entry[0].Entry_Offset = 0;
/*********************************************************************/
/*  This sets up the find option list.                               */
/*********************************************************************/
  fnd_option_list->Rule = _Equals;
  fnd_option_list->Occ_Count = 100;
/*********************************************************************/
/*  This establishes a linkage to a C/400 program which does I/O     */
/*  operations.                                                      */
/*********************************************************************/
clibinit;
/*********************************************************************/
/*  This loops until the choice 'Q' is entered at the menu.          */
/*********************************************************************/
 while (1==1) {
  printf("\n\n**********************\n");
  printf("*   TELEPHONE INDEX   *\n");
  printf("**********************\n");
  printf("* 'A'  Add name & num *\n");
  printf("* 'L'  List a number  *\n");
  printf("* 'Q'  Quit index     *\n");
  printf("**********************\n");
  gets(response);
  if ((strncmp(response,"A",1)==0)||(strncmp(response,"a",1)==0))
   { printf("\nEnter name to add. ex(Last, First)\n");
     gets(name);
     convert_case(name);
     printf("\nEnter number to add. ex(999-9999)\n");
     gets(number);
     strcpy(name,strcat(name,"     "));
```

```
    strcpy(Name_And_Num,strcat(name,number));
    printf("\nName and number to add is => %s\n",Name_And_Num);
    insinxen(index,Name_And_Num,&ins_option_list);
  } /* if 'a'*/
 if ((strncmp(response,"L",1)==0)||(strncmp(response,"l",1)==0))
  {
    printf("\nEnter name to find. ex(Last, First)\n");
    gets(In_Name);
    convert_case(In_Name);
    fnd_option_list->Arg_Length = strlen(In_Name);
    fndinxen(Out_Num,index,fnd_option_list,In_Name);
    length_of_entry = fnd_option_list->Entry[0].Entry_Length;
    Ent_Found = fnd_option_list->Ret_Count;
    if (Ent_Found == 0)
      printf("\nName not found in index => %s\n",In_Name);
    else {
      if (Ent_Found > 1)  {
       printf("\n%d occurrences found,\n",Ent_Found);
       count = 0;
       start = 0;
       while (count++ < Ent_Found) {
        printf("Name and number is => %s\n",Out_Num);
        start = start + length_of_entry;
        strncpyn(Out_Num,Out_Num,start,length_of_entry);
       } /* while */
      }else
       printf("\nName and number is => %s\n",Out_Num);
    }  /*else*/
  }  /*if 'l'*/
 if ((strncmp(response,"Q",1)==0)||(strncmp(response,"q",1)==0))
  { break; }
 }  /*while*/
clibterm;
} /*$USIDXEX*/
```

To create the System C/400 PRPQ program to insert entries into the user index, specify the following:

```
CRTSYSCTXT TXT(QGPL/$USIDXEX) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/$USIDXEX) FROMTXT(QGPL/$USIDXEX)
```

Creating and Manipulating a User Index

The following example shows how to create and manipulate a user index with a call from an MI program. For another example using the QUSCRTUI API, see "Creating Your Own Telephone Directory" on page A-14.

```
/*****************************************************************/
/*                                                               */
/* PROGRAM:  GLOBALV                                             */
/*                                                               */
/* LANGUAGE:  MI/IRP                                             */
/*                                                               */
/* DESCRIPTION:  MAINTAINS AN INDEPENDENT INDEX.  EACH INDEX ENTRY */
/*               CONTAINS 100 BYTES OF USER DATA.  THE ENTRIES ARE */
/*               KEYED TWO 10 BYTE VALUES:  THE USER PROFILE AND A */
/*               VALUE IDENTIFIER.                               */
/*                                                               */
/* APIs USED:  QUSCRTUI                                          */
/*                                                               */
/* PARAMETERS:                                                   */
/*                                                               */
/*   PARM  TYPE       DESCRIPTION                                */
/*                                                               */
/*    1    CHAR(1)    FUNCTION:                                  */
/*                                                               */
/*                    'U':  UPDATE GLOBALV INFORMATION           */
/*                    'R':  RETRIEVE GLOBALV INFORMATION         */
/*                                                               */
/*    2    CHAR(10)   USER PROFILE                               */
/*                                                               */
/*                    THE NAME OF THE USER PROFILE FOR WHICH     */
/*                    INFORMATION IS TO BE SAVED OR RETRIEVED.   */
/*                                                               */
/*    3    CHAR(10)   VALUE ID                                   */
/*                                                               */
/*                    THE NAME OF THE GLOBALV VARIABLE ID FOR WHICH */
/*                    INFORMATION IS TO BE SAVED OR RETRIEVED.   */
/*                                                               */
/*    4    CHAR(100)  VALUE                                      */
/*                                                               */
/*                    IF FUNCTION IS 'U', THIS VALUE SHOULD CONTAIN */
/*                    THE NEW VALUE TO BE ASSOCIATED WITH THE    */
/*                    USER ID AND VALUE ID.                      */
/*                                                               */
/*                    IF FUNCTION IS 'R', THIS VARIABLE WILL BE  */
/*                    SET TO THE VALUE ASSOCIATED WITH THE USER ID */
/*                    AND VALUE ID.  IF NO VALUE EXISTS, *NONE   */
/*                    IS SPECIFIED.                              */
/*                                                               */
/*****************************************************************/

ENTRY * (GLOBALV_PARM) EXT;

/*****************************************************************/
/* PARAMETER VALUE POINTERS FOR GLOBALV.                         */
/*****************************************************************/

DCL SPCPTR GV_REQUEST@ PARM;
DCL SPCPTR GV_USERID@ PARM;
DCL SPCPTR GV_VALUEID@ PARM;
DCL SPCPTR GV_VALUE@ PARM;

/*****************************************************************/
/* PARAMETER VALUES FOR GLOBALV.                                 */
/*****************************************************************/

DCL DD GV_REQUEST CHAR(1) BAS(GV_REQUEST@);
DCL DD GV_USERID CHAR(10) BAS(GV_USERID@);
DCL DD GV_VALUEID CHAR(10) BAS(GV_VALUEID@);
DCL DD GV_VALUE CHAR(100) BAS(GV_VALUE@);

/*****************************************************************/
/* PARAMETER LIST FOR GLOBALV.                                   */
/*****************************************************************/

DCL OL GLOBALV_PARM (GV_REQUEST@
                    ,GV_USERID@
                    ,GV_VALUEID@
                    ,GV_VALUE@
                    ) PARM EXT;

/*****************************************************************/
/* ARGUMENT VALUES FOR CREATE USER INDEX (QUSCRTUI) API.         */
/*****************************************************************/

DCL DD UI_NAME CHAR(20) INIT("GLOBALV   QGPL      ");
DCL DD UI_ATTR CHAR(10) INIT("          ");
DCL DD UI_EATR CHAR(1) INIT("F");
DCL DD UI_ELEN BIN(4) INIT(120);
DCL DD UI_KATR CHAR(1) INIT("1");
DCL DD UI_KLEN BIN(4) INIT(20);
DCL DD UI_IUPD CHAR(1) INIT("0");
DCL DD UI_OPT  CHAR(1) INIT("0");
DCL DD UI_AUT  CHAR(10) INIT("*CHANGE   ");
DCL DD UI_TEXT CHAR(50)
       INIT("GLOBALV INDEX                                    ");

/*****************************************************************/
/* POINTERS TO ARGUMENT VALUES FOR QUSCRTUI API.                 */
/*****************************************************************/

DCL SPCPTR UI_NAME@ INIT(UI_NAME);
DCL SPCPTR UI_ATTR@ INIT(UI_ATTR);
DCL SPCPTR UI_EATR@ INIT(UI_EATR);
DCL SPCPTR UI_ELEN@ INIT(UI_ELEN);
DCL SPCPTR UI_KATR@ INIT(UI_KATR);
DCL SPCPTR UI_KLEN@ INIT(UI_KLEN);
DCL SPCPTR UI_IUPD@ INIT(UI_IUPD);
DCL SPCPTR UI_OPT@  INIT(UI_OPT);
DCL SPCPTR UI_AUT@  INIT(UI_AUT);
DCL SPCPTR UI_TEXT@ INIT(UI_TEXT);

/*****************************************************************/
/* ARGUMENT LIST FOR QUSCRTUI API.                               */
/*****************************************************************/

DCL OL QUSCRTUI_ARG (UI_NAME@
                    ,UI_ATTR@
                    ,UI_EATR@
                    ,UI_ELEN@
```

```
                  ,UI_KATR@
                  ,UI_KLEN@
                  ,UI_IUPD@
                  ,UI_OPT@
                  ,UI_AUT@
                  ,UI_TEXT@
                  ) ARG;

/*****************************************************************/
/* SYTSEM POINTER TO QUSCRTUI API *PGM OBJECT.                   */
/*****************************************************************/

DCL SYSPTR QUSCRTUI INIT("QUSCRTUI",TYPE(PGM));

/*****************************************************************/
/* SYSTEM POINTER TO GLOBALV *USRIDX OBJECT.                     */
/*****************************************************************/

DCL SYSPTR INX@;

DCL DD INX_OBJECTID CHAR(34);
DCL DD INX_OBJECTID_TYPE CHAR(2) DEF(INX_OBJECTID) POS(1)
       INIT(X'0E0A');
DCL DD INX_OBJECTID_NAME CHAR(30) DEF(INX_OBJECTID) POS(3)
       INIT('GLOBALV                       ');
DCL DD INX_OBJECTID_AUT CHAR(2) DEF(INX_OBJECTID) POS(33)
       INIT(X'0000');

/*****************************************************************/
/* EXCEPTION MONITOR TO DETECT 2201X EXCEPTIONS (OBJECT NOT FOUND)  */
/*****************************************************************/

DCL EXCM EXCM_NOOBJECT EXCID(H"2201") INT(CREATE_INDEX) IMD;

/*****************************************************************/
/* PASA INVOCATION ENTRY FOR RETURN FROM EXCEPTION.              */
/*****************************************************************/

DCL DD RTN_NOOBJECT CHAR(18) BDRY(16);
DCL SPCPTR RTN_NOOBJECT@ INIT(RTN_NOOBJECT);
DCL DD RTN_NOOBJECT_ADDR CHAR(16) DEF(RTN_NOOBJECT);
DCL DD RTN_NOOBJECT_OPT CHAR(1) DEF(RTN_NOOBJECT) POS(18)
       INIT(X'00');

/*****************************************************************/
/* RECEIVER VARIABLE FOR INDEPENDENT INDEX OPERATIONS.           */
/*****************************************************************/

DCL DD INX_RECEIVER CHAR(120);
DCL SPCPTR INX_RECEIVER@ INIT(INX_RECEIVER);

/*****************************************************************/
/* OPTION TEMPLATE FOR INDEPENDENT INDEX OPERATIONS.             */
/*****************************************************************/

DCL DD INX_OPT CHAR(14);
DCL SPCPTR INX_OPT@ INIT(INX_OPT);
DCL SPC INX_OPT_SPC BAS(INX_OPT@);
DCL DD INX_OPT_RULE CHAR(2) DIR;
DCL DD INX_OPT_ARGL BIN(2) DIR;
DCL DD INX_OPT_ARGO BIN(2) DIR;
DCL DD INX_OPT_OCCC BIN(2) DIR;
DCL DD INX_OPT_RTNC BIN(2) DIR;
DCL DD INX_OPT_ELEN BIN(2) DIR;
DCL DD INX_OPT_EOFF BIN(2) DIR;

/*****************************************************************/
/* ARGUMENT VARIABLE FOR INDEPENDENT INDEX OPERATIONS.           */
/*****************************************************************/

DCL DD INX_ARG CHAR(120);
DCL SPCPTR INX_ARG@ INIT(INX_ARG);

/*****************************************************************/
/* START OF CODE                                                 */
/*****************************************************************/

  MATINVE RTN_NOOBJECT_ADDR,*,X'03'; /* MATERIALIZE THIS PROGRAM'S  */
                                     /* INVOCATION ENTRY IN THE     */
                                     /* PASA.  THIS ENTRY IS USED   */
                                     /* WHEN RETURNING FROM THE     */
                                     /* EXCEPTION HANDLER BELOW.    */

  RSLVSP INX@,INX_OBJECTID,*,*; /* RESOLVE TO "GLOBALV" USER INDEX  */
                                /* OBJECT.  IF THE OBJECT DOES NOT  */
                                /* EXIST, THEN THE X'2201' EXCEPTION*/
                                /* IS RETURNED, CAUSING THE "OBJECT */
                                /* NOT FOUND" EXCEPTION HANDLER AT  */
                                /* THE END OF THE PROGRAM TO RUN.   */


  CMPBLA(B) GV_REQUEST,'U'/NEQ(NOT_UPDATE); /* IF GV_REQUEST ¬= U   */
                                            /* BRANCH TO NOT_UPDATE */

    /* SET UP OPTIONS FOR INSERT INDEPENDENT INDEX ENTRY (INSINXEN) */
    /* OPERATION.                                                   */

    CPYBLA INX_OPT_RULE,X'0002';     /* RULE= INSERT.               */
    CPYNV  INX_OPT_OCCC,1;           /* OCCURENCE COUNT = 1.        */
    CPYBLA INX_ARG(1:10),GV_USERID;  /* SPECIFY INDEX ENTRY.        */
    CPYBLA INX_ARG(11:10),GV_VALUEID;
    CPYBLA INX_ARG(21:100),GV_VALUE;
    INSINXEN INX@,INX_ARG@,INX_OPT@; /* INSERT THE INDEX ENTRY.     */
    RTX *;                           /* RETURN                      */

NOT_UPDATE:

  CMPBLA(B) GV_REQUEST,'R'/NEQ(NOT_RETRIEVE); /* IF GV_REQUEST ¬= R */
                                              /* GOTO NOT_RETRIEVE. */

    /* SET UP OPTIONS FOR FIND INDEPENDENT INDEX ENTRY (FNDINXEN)   */
    /* OPERATION.                                                   */

    CPYBLA INX_OPT_RULE,X'0001';    /* RULE= FIND WITH EQUAL KEY.   */
    CPYNV  INX_OPT_ARGL,20;         /* ARGUMENT LENGTH= 20.         */
    CPYNV  INX_OPT_OCCC,1;          /* OCCURRENCE COUNT=1.          */
    CPYBLA INX_ARG(1:10),GV_USERID; /* SPECIFY SEARCH ARGUMENT.     */
    CPYBLA INX_ARG(11:10),GV_VALUEID;
    FNDINXEN INX_RECEIVER@,INX@,INX_OPT@,INX_ARG@; /* FIND ENTRY.   */
    CMPNV(B) INX_OPT_RTNC,1/EQ(FOUND_ENTRY); /* IF RETURN_COUNT = 1 */
                                             /* GOTO FOUND_ENTRY.   */
      CPYBLAP GV_VALUE,'*NONE',' '; /* ENTRY WAS NOT FOUND, SPECIFY */
                                    /* VALUE OF *NONE.              */
      RTX *;                        /* RETURN                       */

FOUND_ENTRY:

      CPYBLA GV_VALUE,INX_RECEIVER(21:100); /* ENTRY WAS FOUND,     */
                                            /* COPY VALUE TO USER   */
                                            /* PARAMETER.           */
      RTX *;                                /* RETURN               */

NOT_RETRIEVE:

  RTX *; /* UNKNOWN FUNCTION CODE.  RETURN.                         */

/*****************************************************************/
/* "OBJECT NOT FOUND" EXCEPTION HANDLER.                         */
/*****************************************************************/

ENTRY CREATE_INDEX INT;

  MODEXCPD EXCM_NOOBJECT,X'0000',X'01'; /* TURN OFF EXCEPTION      */
                                        /* MONITOR.                */

  CALLX QUSCRTUI,QUSCRTUI_ARG,*; /* USE QUSCRTUI API TO CREATE THE  */
                                 /* USER INDEX OBJECT.              */

  RTNEXCP RTN_NOOBJECT@; /* RETURN FROM THE EXCEPTION HANDLER AND   */
                         /* RETRY THE OPERATION.                    */
PEND;
```

Creating a Batch Machine

These System C/400 PRPQ programs emulate a batch machine. One program, $USQEXSRV, acts as a server and takes the entries off a user queue and then runs the request through the Execute Command API, QCMDEXC. (The QCMDEXC API is described in the *CL Programmer's Guide*.) The other program, $USQEXREQ, acts as a requester and

puts the entries into a user space. The APIs used in this example are:

- Create User Queue (QUSCRTUQ)
- Execute Command (QCMDEXC)

Requester Program ($USQEXREQ)

The following is a requester program using System C/400 PRPQ:

```
/*****************************************************************/
/*  PROGRAM:  $USQEXREQ                                          */
/*                                                               */
/*  LANGUAGE:  SYSTEM C/400 PRPQ                                 */
/*                                                               */
/*  DESCRIPTION:  This program enters commands to be run onto a  */
/*    user queue called TESTQ in library QGPL.  The user is      */
/*    prompted to enter as many commands (under 51 characters) as*/
/*    necessary.  To end the programs, enter QUIT at the prompt. */
/*                                                               */
/*  APIs USED:  None                                             */
/*                                                               */
/*****************************************************************/
#define C400
#include <string.h>    /* strcpy, strncmp                */
#include <miqueue.h>   /* _ENQ_msg_prefix, enq           */
#include <miptrnam.h>  /* _SYSPTR, rslvsp                */
#include "C400.h"      /* clibinit, clibterm, gets, printf */
                       /* _Usrq, _AUTH_ALL               */


main()
{
_ENQ_msg_prefix e_msg_prefix;
_SYSPTR queue;
char INMsg[100];
/*****************************************************************/
/*  This resolves to the user queue created by $USQEXSRV.        */
/*****************************************************************/
queue = rslvsp(_Usrq,"TESTQ","QGPL",_AUTH_ALL);
e_msg_prefix.msg_len = 100;
/*****************************************************************/
/*  This establishes a linkage to a C/400 program which does I/O */
/*  operations.                                                  */
/*****************************************************************/
clibinit;
/*****************************************************************/
/*  This loops until the user enters 'quit' as the command.      */
/*****************************************************************/
  while (1==1) {
    printf("\nEnter command to put on queue, or 'quit' \n ");
    gets(INMsg);
    printf("\nCommand entered was ==> %.100s\n",INMsg);
/*****************************************************************/
/*  This checks to see if the user entered 'quit' as the command.*/
/*  If it is true, the 'while' loop is ended.                    */
/*****************************************************************/
    if ((strncmp(INMsg,"quit",4) == 0)||(strncmp(INMsg,"QUIT",4) == 0))
      { break; }
/*****************************************************************/
/*  This adds the user-entered command to the user queue.        */
/*****************************************************************/
    enq(queue,&e_msg_prefix,INMsg);
    strcpy(INMsg," ");
  } /*while*/
/*****************************************************************/
/*  This adds the command END to the user queue which causes the */
/*  end of the server program ($USQEXSRV).                       */
/*****************************************************************/
  strcpy(INMsg,"END");
  enq(queue,&e_msg_prefix,INMsg);
clibterm;
} /* $USQEXREQ */
```

To create the requester program using System C/400 PRPQ, specify the following:

```
CRTSYSCTXT TXT(QGPL/$USQEXREQ) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/$USQEXREQ) FROMTXT(QGPL/$USQEXREQ)
```

Server Program ($USQEXSRV)

The following is the server program using System C/400 PRPQ:

```
/*****************************************************************/
/*  PROGRAM:  $USQEXSRV                                          */
/*                                                               */
/*  LANGUAGE:  System C/400 PRPQ                                 */
/*                                                               */
/*  DESCRIPTION:  This program removes commands that are to be run */
/*    from a user queue called 'TESTQ' in library 'QGPL'.  The   */
/*    commands are removed and run in FIFO order.  The user queue */
/*    is created before its use and should be deleted after you use */
/*    this example program.  This program ends when it gets the  */
/*    command 'END' from the user queue.                         */
/*       The flow is as follows:                                 */
/*         (1)  Create the user queue.                           */
/*         (2)  Enter the loop                                   */
/*         (3)     Wait forever for a command on the user queue. */
/*         (4)      If the command END exists, end the loop.     */
/*         (5)      Or else run the command and restart the loop. */
/*         (6)  End the loop                                     */
/*    For the best results, the user should call this program and */
/*    call $USQEXREQ from another session.                       */
/*                                                               */
/* APIs USED:  QCMDEXC, QUSCRTUQ                                 */
/*                                                               */
/*****************************************************************/
#define C400
#include <string.h>     /* strlen, strncmp            */
#include <micomput.h>   /* _DPA_Template_T, cvtnci    */
#include <miqueue.h>    /* _DEQ_msg_prefix, deq       */
#include <miptrnam.h>   /* _SYSPTR, rslvsp            */
#include "OPUSAPI.h"    /* Linkage info, structures   */
#include "C400.h"       /* _Usrq, _AUTH_ALL           */


main()
{

/*****************************************************************/
/*    This sets up the interface to the external QCMDEXC API     */
/*    to run the commands extracted from the user queue.       */
/*****************************************************************/
#pragma linkage(QCMDEXC,OS)
  void QCMDEXC (char comm_to_exec[],
                char length_of_call[15],
                char third_param[3]);

_DEQ_msg_prefix d_msg_prefix;
_DPA_Template_T d_tmp_t;
_SYSPTR queue;
char OUTMsg[100];
int cmd_name_lngth;
char pack_name_lngth[15];
char igc_param[] = "IGC";
/*****************************************************************/
/*    This sets up the parameters to call to the QUSCRTUQ API.   */
/*****************************************************************/
char q_name[]= "TESTQ     QGPL      ";
char ext_atr[]= "TESTER    ";
char q_type[]= "F";
int key_lngth = 0;
int max_msg_s = 100;
int int_msgs  = 10;
int add_msgs = 50;
char auth[]  = "*ALL      ";
char desc[] = "Description .....    ";
/*****************************************************************/
/*  This calls the QUSCRTUQ API to create the user queue.        */
/*****************************************************************/
  QUSCRTUQ(q_name,ext_atr,q_type,key_lngth,max_msg_s,int_msgs,
           add_msgs,auth,desc);
/*****************************************************************/
/*  This resolves to the created user queue.                     */
/*****************************************************************/
  queue = rslvsp(_Usrq,"TESTQ","QGPL",_AUTH_ALL);
/*****************************************************************/
/*  This sets the dequeue operation to wait for the command forever.*/
/*****************************************************************/
```

```
  d_msg_prefix.wait_forever = 1;
  d_tmp_t.Type = T_Packed;
  d_tmp_t.Length = 0x050F;
  d_tmp_t.reserved = 0x00000000;
/*********************************************************************/
/*  This loops until the command 'END' is removed from the user     */
/*  queue.                                                           */
/*********************************************************************/
  while (1==1) {
    deq(&d_msg_prefix,OUTMsg,queue);
/*********************************************************************/
/*  This checks to see if the command 'END' is removed.  If this is */
/*  true, then end the 'while' loop.                                 */
/*********************************************************************/
    if (strncmp(OUTMsg,"END",3) == 0)
      { break; }
    cmd_name_lngth = strlen(OUTMsg);
/*********************************************************************/
/*  This converts the integer in cmd_name_lngth to a packed decimal */
/*  using arguments in d_tmp_t.                                      */
/*********************************************************************/
    cvtnci(pack_name_lngth,cmd_name_lngth,d_tmp_t);
/*********************************************************************/
/*  This runs the command removed from the user queue.             */
/*********************************************************************/
    QCMDEXC(OUTMsg,pack_name_lngth,igc_param);
  } /* while */
}  /* $USQEXSRV */
```

To create the server program using System C/400 PRPQ, specify the following:

```
CRTSYSCTXT TXT(QGPL/$USQEXSRV) SRCFILE(QGPL/QCSRC)
CRTSYSCPGM PGM(QGPL/$USQEXSRV) FROMTXT(QGPL/$USQEXSRV)
```

Using the Create Program (QPRCRTPG) API

The following PL/I program shows how the QPRCRTPG API can be used to create a program:

```
/*********************************************************************/
/*                                                                   */
/* MODULE: QPRUSRP                                                   */
/*                                                                   */
/* LANGUAGE:  PL/I                                                   */
/*                                                                   */
/* FUNCTION: Creates a program using the QPRCRTPG API.  Reads the    */
/*           intermediate representation of program to be used from  */
/*           a source file member.                                   */
/*                                                                   */
/* APIs used: QPRCRTPG                                               */
/*                                                                   */
/*********************************************************************/
QPRUSRP: PROCEDURE (PGM, TEXT, SRCF, SRCM, SRCDT, AUT, OPTIONS);

DECLARE
  PGM CHAR(20),                 /* The first ten characters        */
                                /* specifies the name of the       */
                                /* program to be created.  The     */
                                /* second ten characters specifies */
                                /* the library to be used.         */

  TEXT CHAR(50),                /* Contains a brief description of */
                                /* the program.                    */

  SRCF CHAR(20),                /* The first ten characters        */
                                /* specify the name of the source  */
                                /* file containing the intermediate*/
                                /* representation of program. The  */
                                /* second ten characters specify   */
                                /* the library containing the      */
                                /* source file.                    */

  SRCM CHAR(10),                /* Specifies the source file       */
                                /* member containing the           */
                                /* intermediate representation of  */
                                /* program.                        */

  SRCDT CHAR(13),               /* Specifies the last changed date */
                                /* and time for the source file    */
                                /* member.                         */

  AUT CHAR(10),                 /* Specifies the authority to be   */
                                /* used.                           */

  OPTIONS CHAR(176);            /* Specifies the option values to  */
                                /* be used.  These are stored as a */
                                /* series of CHAR(11) values.      */

DECLARE
  IRPRGM FILE;                  /* The source file containing the  */
                                /* intermediate representation of  */
                                /* program.                        */

DECLARE
  QPRCRTPG ENTRY(*, *, *, *, *, *, *, *, *, *, *, *, *)
           OPTIONS(ASSEMBLER);
                                /* This declares the QPRCRTPG API. */

DECLARE
  INPUT_BUFFER CHAR(92),        /* Used to read data from the      */
                                /* file.  The first 12 bytes       */
                                /* contain the time/date stamp.    */
                                /* The remaining 80 bytes contain  */
                                /* source data.                    */

  INPUT_ARRAY(32000) CHAR(80),  /* Used to store the intermediate  */
                                /* representation of program.  Up  */
                                /* to 32000 records may be         */
                                /* processed.                      */

  INPUT_SIZE BINARY(31),        /* Contains the number of input    */
                                /* records processed.              */

  PAGE_NUM BINARY(31) STATIC INIT(1),
                                /* Specifies the starting page     */
                                /* number.                         */

  NUM_OPTIONS BINARY(31) STATIC INIT(16);
                                /* Specifies the number of option  */
                                /* values in the option template.  */

DECLARE
```

```
    1 BIT_FLAGS STATIC,
      3 MORE_RECORDS BIT(1) ALIGNED,
      3 NO           BIT(1) ALIGNED INIT('0'),
      3 YES          BIT(1) ALIGNED INIT('1');

  ON ENDFILE(IRPRGM)
    MORE_RECORDS= NO;

    OPEN FILE(IRPRGM);

    MORE_RECORDS= YES;
    INPUT_SIZE= 0;
    GET FILE(IRPRGM) EDIT(INPUT_BUFFER)(A(92));
    DO WHILE(MORE_RECORDS);
      INPUT_SIZE= INPUT_SIZE + 1;
      INPUT_ARRAY(INPUT_SIZE)= SUBSTR(INPUT_BUFFER, 13, 80);
      GET FILE(IRPRGM) EDIT(INPUT_BUFFER)(A(92));
    END; /* DO WHILE */

    IF INPUT_SIZE > 0 THEN
    DO;
      CALL QPRCRTPG (INPUT_ARRAY
                    ,INPUT_SIZE * 81
                    ,PGM
                    ,TEXT
                    ,SRCF
                    ,SRCM
                    ,SRCDT
                    ,'QSYSPRT   *LIBL     '
                    ,PAGE_NUM
                    ,AUT
                    ,OPTIONS
                    ,NUM_OPTIONS
                    );
    END; /* IF */

    CLOSE FILE(IRPRGM);
  END QPRUSRP;
```

Using Profile Handles

The following example illustrates how to generate, change, and release profile handles in a CL program. The example uses three of the security APIs:

- Get Profile Handle (QSYGETPH)
- Set Profile (QWTSETP)
- Release Profile Handle (QSYRLSPH)

```
PGM (&USERID &PWD)

/* Declare the variables needed by this program:           */
DCL        VAR(&USERID) TYPE(*CHAR) LEN(10)
DCL        VAR(&PWD) TYPE(*CHAR) LEN(10)
DCL        VAR(&SECOFR) TYPE(*CHAR) LEN(10) VALUE(QSECOFR)
DCL        VAR(&SECPWD) TYPE(*CHAR) LEN(10) VALUE(*NOPWD)
DCL        VAR(&PRFHNDL1) TYPE(*CHAR) LEN(12)
DCL        VAR(&PRFHNDL2) TYPE(*CHAR) LEN(12)

/* Generate profile handles for the QSECOFR user ID and    */
/* for the user ID passed to this program:                 */

CALL       PGM(QSYGETPH) PARM(&SECOFR &SECPWD &PRFHNDL1)
CALL       PGM(QSYGETPH) PARM(&USERID &PWD &PRFHNDL2)

| /* Authorize the user ID passed to this program to the    */
| /* QWTSETP API.                                          */

| GRTOBJAUT OBJ(QWTSETP) OBJTYPE(*PGM) USER(&USERID) AUT(*USE)

| /* Change the user for this job to the user ID passed to  */
| /* this program:                                         */

CALL       PGM(QWTSETP) PARM(&PRFHNDL2)

/* This program is now running under the user ID passed to */
/* this program.                                           */

/* Now change the user ID for this job back to the QSECOFR */
/* user ID:                                                */

CALL       PGM(QWTSETP) PARM(&PRFHNDL1)

| /* Revoke authority to the QWTSETP API for the user ID    */
| /* passed to this program.                               */

| RVKOBJAUT OBJ(QWTSETP) OBJTYPE(*PGM) USER(&USERID) AUT(*USE)

/* The profile handles generated in this program can now   */
/* be released:                                            */

CALL       PGM(QSYRLSPH) PARM(&PRFHNDL1)
CALL       PGM(QSYRLSPH) PARM(&PRFHNDL2)

ENDPGM
```

Generating and Sending an Alert

The following Pascal program uses both alert APIs. First, it calls the Generate Alert API, QALGENA, to generate an alert without sending a message to the QSYSOPR or QHST message queue. Then it uses the Send Alert API, QALSNDA, to send the alert to the OS/400 alert manager for further processing.

```
program example;

type
  (* ********************************** *)
  (* Simple string types.               *)
  (* ********************************** *)
  string07  = packed array(.1..7.)   of char;
  string10  = packed array(.1..10.)  of char;
  string512 = packed array(.1..512.) of char;
  string100 = packed array(.1..100.) of char;

  (* ********************************** *)
  (* Alert table record.  This is also the *)
  (* message file for the message ID.      *)
  (* ********************************** *)
  alrtbl = packed record
    objname : string10;
    libname : string10;
  end;

  (* ********************************** *)
  (* Error code  structure.  The BYTESPROV *)
  (* field is input; the rest are output.  *)
  (* ********************************** *)
  errorcode = packed record
    bytesprov     : integer;
    bytesavail    : integer;
    messageid     : string07;
    reserved      : char;
    exceptiondata : string100;
  end;

(* ************************************************** *)
(* This is the application program interface to create an *)
(* alert when given a message ID and alert table.         *)
(*                                                        *)
(* The parameters passed are:                             *)
(*    ALERT      - O   - The  alert  major vector that is *)
(*                       created                          *)
(*    PROVIDED   - I   - The number of bytes of  provided *)
(*                       space                            *)
(*    NEEDED     - O   - The  number  of  bytes of needed *)
(*                       space                            *)
(*    ALERTTBL   - I   - The alert table and message file *)
(*                       containing the alert description *)
(*                       and message description          *)
(*    MESSAGE    - I   - The message ID to use            *)
(*    MSGDATA    - I   - The  message  substitution  text *)
(*    MSGDATALEN - I   - The number of bytes  provided in *)
(*                       MSGDATA                          *)
(*    RETURNCODE - I/O - The error code structure         *)
```

```
(* ************************************************** *)
procedure qalgena(
   var alert      : string512;
   var provided   : integer;
   var needed     : integer;
   var alerttbl   : alrtbl;
   var message    : string07;
   var msgdata    : string100;
   var msgdatalen : integer;
   var returncode : errorcode); nonpascal;

(* ************************************************** *)
(* This is the application program interface to send an   *)
(* alert to the alert manager for further processing.     *)
(*                                                        *)
(* The parameters passed are:                             *)
(*    ALERT      - I   - The alert major vector that is   *)
(*                       sent                             *)
(*    PROVIDED   - I   - Number of bytes provided by the  *)
(*                       alert major vector               *)
(*    LCLRCVIND  - I   - An indicator of a local or       *)
(*                       received alert                   *)
(*    ORIGIN     - I   - The origin of the alert          *)
(*    RETURNCODE - I/O - The error code structure         *)
(* ************************************************** *)
procedure qalsnda(
   var alert      : string512;
   var provided   : integer;
   var lclrcvind  : char;
   var origin     : string10;
   var returncode : errorcode); nonpascal;

var
  (* ************************************ *)
  (* Local copies of the parameters       *)
  (* ************************************ *)
   alert     : string512;         (* The alert major vector space    *)
   bytespass : integer;           (* The number of bytes passed in   *)
   bytesrtn  : integer;           (* The number of bytes returned    *)
   table     : alrtbl;            (* The alert table/message file    *)
   msgid     : string07;          (* The message ID to use           *)
   msgdata   : string100;         (* The message data to substitute  *)
   msglength : integer;           (* The length of the message data  *)
   localrecd : char;              (* The local/received indicator    *)
   origin    : string10;          (* The origin of the alert to send *)
   rtncode   : errorcode;         (* The return code for the APIs    *)

begin

   (* Start by generating an alert for a specific message *)
   alert             := ' ';              (* Initialize alert space   *)
   bytespass         := length(alert);    (* Initialize in length     *)
   bytesrtn          := 0;                (* Initialize out length    *)
   msgid             := 'CPA2601';        (* Initialize message ID    *)
   table.objname     := 'QCPFMSG   ';     (* Initialize table name    *)
   table.libname     := 'QSYS      ';     (* Initialize table library *)
   msgdata           := ' ';              (* Initialize data          *)
   msglength         := 0;                (* Initialize data length   *)
   rtncode.bytesprov := 116;              (* Initialize RC length     *)
   rtncode.messageid := '       ';       (* Initialize RC message    *)
   qalgena(alert,
           bytespass,
           bytesrtn,
           table,
           msgid,
           msgdata,
           msglength,
           rtncode);                      (* Generate the alert       *)

   (* Check the return code from QALGENA *)
   if rtncode.messageid = '       ' then
      (* Return code is OK, so send the generated alert *)
      begin                                        (* Alert OK          *)
         localrecd             := 'L';             (* Local alert       *)
         origin                := 'EXAMPLE   ';    (* From EXAMPLE pgm  *)
         rtncode.bytesprov     := 116;             (* Init RC length    *)
         rtncode.messageid     := '       ';       (* Blank RC message  *)
         rtncode.exceptiondata := ' ';             (* Blank RC data     *)
         qalsnda(alert,
                 bytesrtn,
                 localrecd,
                 origin,
                 rtncode);                         (* Send the alert    *)
      end                                          (* ...Alert OK       *)
   else
      (* QALGENA failed, so print out the exception ID *)
      begin;                                       (* QALGENA failed    *)
         writeln('QALGENA had an error');          (* Tell user about   *)
         write('The error was ');                  (*  the error and    *)
         writeln(rtncode.messageid);               (*  the exception ID *)
      end;                                         (* ...QALGENA failed *)

end.
```

Diagnostic Reporting

The following example program illustrates the use of the Send Nonprogram Message API, QMHSNDM, and the Receive Program Message API, QMHRCVPM. The program produces a diagnostic report of errors that occur when the QMHSNDM API is used to send a message to more than one message queue.

The program calls the QMHSNDM API to send a message to message queues that do not exist. The QMHSNDM API returns a generic exception message, CPF2469. This message indicates that the API also returned one or more diagnostic messages describing the errors. After the program receives the exception message and verifies that it is message CPF2469, it uses the QMHRCVPM API to receive the diagnostic messages. The program prints the exception message, the diagnostic messages, and the message help.

Diagnostic Report (DIAGRPT) Program

```
/*********************************************************************/
/*                                                                   */
/* MODULE NAME:  DIAGRPT - Diagnostic Report                         */
/* LANGUAGE:     C/400                                               */
/*                                                                   */
/* FUNCTION:     This module will produce a diagnostic report that   */
/*               could be used in diagnosing the errors that         */
/*               occurred using the QMHSNDM API to send a message    */
/*               to multiple message queues.                         */
/*               This program purposely causes the QMHSNDM API to    */
/*               try to send a message to message queues that do not*/
/*               exist and as a result the generic CPF2469 exception*/
/*               is returned indicating that 1 or more diagnostic    */
/*               messages were returned identifying the error(s) on */
/*               the send.  The program receives the exception and   */
/*               prints it out, and if it is the CPF2469, it         */
/*               receives the previous diagnostics and also prints   */
/*               them out.                                           */
/*                                                                   */
/* Dependency:   A print file must be created before calling program*/
/*               DIAGRPT.  The print file should be created using    */
/*               the following command-                              */
/*               CRTPRTF FILE(PRTDIAG) CTLCHAR(*FCFC) CHLVAL(1 (13))*/
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <psmhapi.h>

#define  DIAG_TYPE "02"
#define  BUF_SIZE  80

/*********************************************************************/
/* Type definition for error code structure                           */
/*********************************************************************/
typedef struct error_code_struct
  {
   int bytes_provided;
   int bytes_available;
   char exception[7];
```

```
    char RESERVED;
    char exception_data[100];
   } error_code_struct;

/*********************************************************************/
/* Type definition for qualified name structure                      */
/*********************************************************************/
typedef struct qual_name_struct
  {
   char  name[10];
   char  libr[10];
  } qual_name_struct;

/*********************************************************************/
/* Type definition for message information structure used on the     */
/* receive.  F is the fixed portion of the record  and V is the      */
/* variable length portion of the record.                            */
/*********************************************************************/
typedef struct msg_info_struct
  {
   RCVM0200     F;
   char         V[1200];
  } msg_info_struct;

FILE  *prtf;
char  buf[80];
char  received[7];
int   exception_count;


/*********************************************************************/
/* Function to handle errors received on the API calls.              */
/*********************************************************************/
 void error_handler(void)
 {
   sigdata_t *data=sigdata();

   data->sigact->xhalt=0;
   data->sigact->xpmsg=0;
   data->sigact->xumsg=0;
   data->sigact->xdebug=0;
   data->sigact->xdecerr=0;
   data->sigact->xresigprior=0;
   data->sigact->xresigouter=0;
   data->sigact->xremovemsg=0;
   data->sigact->xrtntosgnler=0;
   memcpy(received,data->exmsg->exmsgid,7);
   exception_count++;
   signal(SIGABRT,error_handler);
 }


/*********************************************************************/
/* BuildQList:  Routine to build the message queue list.             */
/*********************************************************************/
void BuildQList(  qual_name_struct *QueueList, int NumQueue)
{
  int i;

  strncpy(QueueList[0].name,"QPGMR     ",10);
  strncpy(QueueList[1].name,"SNOOPY    ",10);
  strncpy(QueueList[2].name,"QSECOFR   ",10);
  strncpy(QueueList[3].name,"PEANUTS   ",10);
  strncpy(QueueList[4].name,"QUSER     ",10);

  for (i = 0; i < NumQueue ; i++ )
    {
     strncpy(QueueList[i].libr,"*LIBL     ",10);
    }
}


/*********************************************************************/
/* PrintError: Routine to print error information & exit.            */
/*********************************************************************/
void PrintError(char *errstring, char exception[7])
{

  memset(buf,' ',BUF_SIZE);
  buf[0] = '0';
  strncpy(buf+1,errstring,strlen(errstring));
  fwrite(buf,1,BUF_SIZE,prtf);
  memset(buf,' ',BUF_SIZE);
  buf[0] = '0';
  strncpy(buf+1,"Exception received->",14);
  strncpy(buf+15,exception,strlen(exception));
  fwrite(buf,1,BUF_SIZE,prtf);
  fclose(prtf);
  exit(1);
 }


/*********************************************************************/
/* PrintData: Routine to print varying length character string data.*/
/*********************************************************************/
void PrintData(char *strname, void *strptr, int strlgth)
{
  char *strdata = strptr;
  int i,lgth,remain;

  /* Write the description and the data that will fit on one line   */
  memset(buf,' ',BUF_SIZE);
  buf[0] = '0';
  lgth = strlen(strname);
  strncpy(buf+1,strname,lgth);
  lgth++;

  /* remain = MIN(strlgth,80 - lgth)  */
  remain = (strlgth < 80 - lgth) ? strlgth : 80 - lgth;
  strncpy(buf+lgth,strdata,remain);
  fwrite(buf,1,BUF_SIZE,prtf);

  /* Now write the remainder of the data */
  if (strlgth > (80 - lgth ))
    {
     /* Adjust pointer to data not printed yet */
     strdata = strdata + (80 - lgth);

     for (i = 0; i < strlgth; i = i + 70, strdata = strdata + 70 )
       {
        /* lgth = MIN(strlgth-i,70)  */
        lgth = (strlgth-i < 70) ? strlgth-i : 70;

        memset(buf,' ',BUF_SIZE);
        strncpy(buf,"0         ",10);
        memcpy(buf+10,strdata,lgth);
        fwrite(buf,1,BUF_SIZE,prtf);
       }

    }
}


/*********************************************************************/
/* PrintMessage: Routine to print the message data & text.           */
/*********************************************************************/
void PrintMessage(msg_info_struct *Msg)

{
  char *DataPtr;     /* Pointer to the varying length character data*/
  int  DataLen;      /* Length of the varying length character data */
  char CharType[10]; /* Message type as a character string          */

  PrintData("Message ID->",Msg->F.msg_id,7);
  /* Convert Message Type to a character string to be printed out   */
  if (memcmp(Msg->F.msg_type,"02",2)==0)
    strncpy(CharType,"DIAGNOSTIC", 10);
  else if (memcmp(Msg->F.msg_type,"15",2)==0)
    strncpy(CharType,"ESCAPE    ",10);
  PrintData("Message Type->",CharType,10);

  /* First point to the beginning of the message data               */
  /* in the structure and get the length of data returned.          */
  DataPtr = Msg->V;
  DataLen = Msg->F.data_returned;
  /* If there is non-blank data, print it out                       */
  if ((DataLen > 0) && (strspn(DataPtr," ") < DataLen))
    PrintData("Message data received->",DataPtr,DataLen);

  /* Point to the beginning of the message text field and get the   */
  /* length of message text returned.                               */
  DataPtr += DataLen;
  DataLen = Msg->F.text1_returned;
  /* If there is non-blank text, print it out                       */
  if ((DataLen > 0) && (strspn( DataPtr," ") < DataLen))
```

```
    PrintData("Message text received->",DataPtr,DataLen);

  /* Now update to point to the beginning of the message           */
  /* help text field and get the length of message help text       */
  /* returned.                                                     */
  DataPtr += DataLen;
  DataLen = Msg->F.text2_returned;
  /* If there is non-blank message help text, print it out         */
  if ((DataLen > 0) && (strspn( DataPtr," ") < DataLen))
    PrintData("Message help text received->",DataPtr,DataLen);
  strncpy(buf,"-                                              ",43);
  fwrite(buf,1,BUF_SIZE,prtf);
}

 /*****************************************************************/
 /*                                                               */
 /*  Start of main program.                                       */
 /*                                                               */
 /*****************************************************************/

main()
{


 error_code_struct ErrorCode;

 qual_name_struct  MsgQList[5];
 qual_name_struct  MsgFile;
 qual_name_struct  RpyMsgQ;

 msg_info_struct   MsgInfo;

 char MsgData[128];
 char MsgText[512];
 char MsgHelp[512];
 char PgmMsgQ[10];
 char MsgType[10];
 char MsgAction[10];
 char Format[8];
 char MsgId[7];
 char MsgKey[4];

 int  MsgTextLen;
 int  MsgInfoLen;
 int  NumMsgQ;
 int  PgmCount;
 int  WaitTime;
 int  morediag;


 /* Initialize variables */
 exception_count = 0;
 signal(SIGABRT,error_handler);
 memcpy(ErrorCode.exception,"       ",7);
 ErrorCode.bytes_provided = 0;

 memcpy(MsgId,"       ",10);
 memcpy(MsgFile.name,"          ",10);
 memcpy(MsgFile.libr,"          ",10);
 strcpy(MsgText,"This is an immediate, informational message");
 MsgTextLen = strlen(MsgText);
 memcpy(MsgType,"*INFO     ",10);
 memcpy(RpyMsgQ.name,"          ",10);
 memcpy(RpyMsgQ.libr,"          ",10);

 /* Build the list of message queues to send the message to       */
 NumMsgQ = 5;
 BuildQList(MsgQList,NumMsgQ);


 /* Send the message to the list of message queues.               */
 QMHSNDM( MsgId,
         &MsgFile,
          MsgText,
          MsgTextLen,
          MsgType,
         &MsgQList,
          NumMsgQ,
         &RpyMsgQ,
         &MsgKey,
         &ErrorCode);

 /* If an error occurred on the send, produce an exception report */
 /* identifying what errors occurred.                             */
 if (exception_count != 0)
   {

    /* Open printer file using first character forms control & write*/
    /* the header information.                                     */
    prtf = fopen ("PRTDIAG", "wb type=record recfm=FA lrecl=80");
    memset(buf,' ',BUF_SIZE);
    strncpy(buf,"1                         DIAGNOSTIC REPORT",43);
    fwrite(buf,1,BUF_SIZE,prtf);
    strncpy(buf,"                          ----------------",43);
    fwrite(buf,1,BUF_SIZE,prtf);
    strncpy(buf,"-                                           ",43);
    fwrite(buf,1,BUF_SIZE,prtf);

    /* Do the setup to first receive the exception signalled.      */
    memcpy(Format,"RCVM0200",8);
    memcpy(PgmMsgQ,"*         ",10);
    memcpy(MsgType,"*EXCP     ",10);
    memcpy(MsgKey,"    ",4);
    memcpy(MsgAction,"*OLD      ",10);
    PgmCount = 0;
    WaitTime = 0;
    MsgInfoLen = 1276;

    /* Now change bytes_provided to 116 so that if any errors occur */
    /* on the receive, the error information will be returned in the*/
    /* error code structure instead of generating more exceptions   */
    /* which will clutter up the program message queue.             */
    ErrorCode.bytes_provided = 116;

    /* Receive the last exception type message on the program      */
    /* message queue                                               */
    QMHRCVPM(&MsgInfo,
             MsgInfoLen,
             Format,
             PgmMsgQ,
             PgmCount,
             MsgType,
             MsgKey,
             WaitTime,
             MsgAction,
            &ErrorCode);

    /* Test for any errors on the receive */
    if (ErrorCode.bytes_available > 0)
      {
       PrintError("QMHRCVPM - Did not complete successfully",
                  ErrorCode.exception);
      }

    /* An exception message was received successfully.  Now see if  */
    /* the message received is the same exception that was signalled*/
    /* If not, there is an error.                                   */
    if (strncmp(MsgInfo.F.msg_id,received,7) != 0)
      {
       PrintError("QMHRCVPM - Wrong exception received",
                  MsgInfo.F.msg_id);
      }


    /* The exception message was received successfully.             */
    /* Print the message data & text for the exception message.     */
    PrintMessage(&MsgInfo);

    /* If the message was the generic CPF2469, there are 1 or more  */
    /* diagnostic messages to go with the CPF2469 on the queue.     */
    /* Receive the diagnostic messages previous to the CPF2469 until*/
    /* a non-diagnostic message is received or there are no more    */
    /* messages.                                                    */
    if (strncmp(MsgInfo.F.msg_id,"CPF2469",7) == 0)
      {
       memcpy(MsgType,"*PRV      ",10);
       memcpy(MsgKey,MsgInfo.F.msg_key,4);
       morediag = 1;

       while(morediag == 1)
         {
          /* Receive the previous diagnostic */
          QMHRCVPM(&MsgInfo,
                   MsgInfoLen,
                   Format,
                   PgmMsgQ,
```

```
                PgmCount,
                MsgType,
                MsgKey,
                WaitTime,
                MsgAction,
                &ErrorCode);

        /* Test for error on the receive */
        if (ErrorCode.bytes_available > 0)
          {
           PrintError("QMHRCVPM - Did not complete successfully",
                      ErrorCode.exception);
          }

        /* If bytes available = 0 OR the next message is not a    */
        /* diagnostic message, we are done.                       */
        if ((MsgInfo.F.bytes_available == 0) ||
            (strncmp(MsgInfo.F.msg_type,DIAG_TYPE,2) != 0) )
          {
           morediag = 0;
          }
        else                /* A diagnostic was received          */
          {
           /* Print the message data & text for the diagnostic msg*/
           PrintMessage(&MsgInfo);

           /* Now copy the message key of the diagnostic message  */
           /* received to the MsgKey parameter to use on the next */
           /* call to QMHRCVPM.                                   */
           memcpy(MsgKey,MsgInfo.F.msg_key,7);
          }

       }                        /* End of while moredia = 1       */

     }                          /* End of if CPF2469 received     */

   /* Write trailer */
   memset(buf,' ',BUF_SIZE);
   strncpy(buf,"-                    END OF DIAGNOSTIC REPORT",48);
   fwrite(buf,1,BUF_SIZE,prtf);

   /* Close the print file */
   fclose(prtf);

  }                             /* End of if error on send        */

}                               /* End mainline                   */
```

Printed Diagnostic Report

The DIAGRPT program produces a report like this:

```
Message ID->CPF2469
Message Type->ESCAPE
Message text received->Error occurred when sending message.
Message help text received->Recovery  . . . :   See messages
          previously listed for a description of the error.
          Correct the error, and then try the
          command again.

Message ID->CPF2403
Message Type->DIAGNOSTIC
Message data received->WILLIAM   *LIBL
Message text received->Message queue WILLIAM in *LIBL not found.
Message help text received->Cause . . . . . :   The message queue you
        specified was not found in the library you specified.  One
        of the following occurred: -- The queue name was not
        entered correctly. -- The queue does not exist in the
        specified library. -- You specified the wrong library name.
        Recovery  . . . :   Do one of the following and try the
        request again: -- Correct or change the message queue
        name or library name used in the message queue (MSGQ)
        parameter or the to-message queue (TOMSGQ) parameter.
        -- Create the message queue using the Create Message
        Queue (CRTMSGQ) command.


Message ID->CPF2403
Message Type->DIAGNOSTIC
Message data received->ANDREW    *LIBL
Message text received->Message queue ANDREW in *LIBL not found.
Message help text received->Cause . . . . . :   The message queue
        you specified was not found in the library you specified.
        One of the following occurred: -- The queue name was not
        entered correctly. -- The queue does not exist in the
        specified library. -- You specified the wrong library
        name. Recovery  . . . :   Do one of the following and
        try the request again: -- Correct or change the message
        queue name or library name used in the message queue
        (MSGQ) parameter or the to-message queue (TOMSGQ)
        parameter. -- Create the message queue using the Create
        Message Queue (CRTMSGQ) command.
```

Listing Directories

To call this program, enter

```
'CALL DIR PARM(/'QDLS')'
```

```
/*********************************************************************/
/*********************************************************************/
/*  FUNCTION:  This program lists a directory to a spooled file.     */
/*             The list resembles the DOS DIR command output.        */
/*                                                                   */
/*  LANGUAGE:  PL/I                                                  */
/*                                                                   */
/*  APIs USED: QHFOPNDR, QHFRDDR, QHFCLODR, QHFLSTFS, QUSCRTUS,      */
/*             QUSRTVUS                                              */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
DIR: PROCEDURE(Dir_Name) OPTIONS(MAIN);

/* parameter declarations                                            */
DCL Dir_Name CHARACTER (100);

/* API entry declarations                                            */
/*                                                                   */
/* The last parameter, the error code, is declared as FIXED BIN(31) */
/* for all APIs.  This always has a value of zero, specifying that  */
/* exceptions should be returned.                                    */
DCL QHFOPNDR ENTRY(CHAR(16),CHAR(100),FIXED BIN(31),CHAR(6),
                   CHAR(52),FIXED BIN(31),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

DCL QHFCLODR ENTRY(CHAR(16),FIXED BIN(31)) OPTIONS(ASSEMBLER);

DCL QHFRDDR ENTRY(CHAR(16),CHAR(200),FIXED BIN(31),FIXED BIN(31),
                  FIXED BIN(31),FIXED BIN(31),FIXED BIN(31))
                  OPTIONS(ASSEMBLER);

DCL QHFLSTFS ENTRY(CHAR (20),CHAR (8),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

DCL QUSCRTUS ENTRY(CHAR (20),CHAR (10),FIXED BIN(31),CHAR (1),
                   CHAR (10),CHAR (50),CHAR (10),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

DCL QUSRTVUS ENTRY(CHAR (20),FIXED BIN(31),FIXED BIN(31),CHAR(*),
                   FIXED BIN(31)) OPTIONS(ASSEMBLER);

DCL QWCCVTDT ENTRY(CHAR (10),CHAR (8),CHAR(10),CHAR(16),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

/*********************************************************************/
/* Parameters for QHFOPNDR                                           */
/*********************************************************************/
/* Directory handle                                                  */
  DCL Dir_Handle CHAR(16);
/* The path name is the Dir_name parameter passed into the           */
/* program.                                                          */
/*                                                                   */
/* Length of path name                                               */
  DCL namelen FIXED BIN(31);
/* Open information                                                  */
  DCL openinfo CHAR(6);
/* Attribute selection table                                         */
  DCL selection CHARACTER(52);           /* selection is used to */
  DCL 1 selection_Struct BASED(Sel_ptr), /* allocate space for   */
          2 num_att   FIXED BIN(31),     /* selction_Struct by   */
          2 offset1   FIXED BIN(31),     /* setting Sel_ptr to   */
```

```
          2 offset2   FIXED BIN(31),      /* the address of       */
          2 offset3   FIXED BIN(31),      /* selection.           */
          2 att_len1  FIXED BIN(31),
          2 att_name1 CHAR(8),
          2 att_len2  FIXED BIN(31),
          2 att_name2 CHAR(8),
          2 att_len3  FIXED BIN(31),
          2 att_name3 CHAR(8);
  DCL Sel_ptr POINTER;
/* Length of attribute selection table                          */
  DCL selectionlen FIXED BIN(31);
/* Error Code                                                   */
  DCL Error_Code FIXED BIN(31);

/*****************************************************************/
/* Parameters for QHFRDDR                                        */
/*****************************************************************/
/* The directory handle is the same as for QHFOPNDR.             */
/*                                                               */
/* Data buffer                                                   */
  DCL buffer CHARACTER(200);            /* buffer is used to     */
  DCL 1 buffer_Struct BASED(Buf_ptr),   /* allocate space for    */
          2 num_dir_ret FIXED BIN(31),  /* buffer_Struct this is */
          2 offset_dir  FIXED BIN(31),  /* done by assigning the */
          2 num_att     FIXED BIN(31),  /* address of buffer to  */
          2 offsets(4)  FIXED BIN(31),  /* Buf_ptr.              */
          2 attinfo     CHARACTER(276);
  DCL Buf_ptr POINTER;
/* The length of the data buffer and number of directory entries */
/* to retrieve are hard coded into the call to QHFRDDR.          */
/*                                                               */
/* Number of directory entries retrieved                         */
  DCL result_count FIXED BIN(31);
/* Length of data returned                                       */
  DCL bytes_returned FIXED BIN(31);
/* The error code is the same as used for QHFOPNDR.              */
/*****************************************************************/

/*****************************************************************/
/* Parameters for QHFCLODR                                       */
/*****************************************************************/
/* The directory handle and error code are the same as used in   */
/* the QHFOPNDR and QHFRDDR APIs.                                */
/*****************************************************************/

/*****************************************************************/
/* Parameters for QUSCRTUS                                       */
/*****************************************************************/
/* The qualified user space name is hard coded into the call to  */
/* QUSCRTUS.                                                     */
/*                                                               */
/* The extended attribute is hard coded into the call.           */
/*                                                               */
/* Initial size                                                  */
  DCL size FIXED BIN(31);
/* The initial value is hard coded into the call.                */
/*                                                               */
/* The public authority is hard coded into the call.             */
/*                                                               */
/* Text description                                              */
  DCL text CHARACTER(50);
/* The replacement option is hard coded into the call.           */
/*                                                               */
/* The error code is the same as for the above APIs.             */
/*****************************************************************/

/*****************************************************************/
/* Parameters for QHFLSTFS                                       */
/*****************************************************************/
/* The qualified user space name and format name are hard coded  */
/* into the call to QHFLSTFS.                                    */
/* The error code is the same as for the above APIs.             */
/*****************************************************************/

/*****************************************************************/
/* Parameters for QUSRTVUS                                       */
/*****************************************************************/
/*                                                               */
/* The qualified user space name is hard coded into the call.    */
/*                                                               */
/* Starting position                                             */
  DCL startpos FIXED BIN(31);
/* Length of data                                                */
  DCL len FIXED BIN(31);
/* Receiver variable                                             */
/*   for binary numbers                                          */
  DCL charbin4 CHARACTER(4);            /* charbin4 allows a     */
  DCL int FIXED BIN(31) BASED(numptr);  /* number to be returned */
  DCL numptr POINTER;                   /* in int by setting     */
                                        /* numptr to the address */
/* NOTE: This same technique is used with charbin4 and int to    */
/* retrieve binary numbers from the attribute information table  */
/* of a directory entry.                                         */
/*                                                               */
/* Receiver variable                                             */
/*   for file system names                                       */
  DCL FSName CHARACTER(10);
/* The error code is the same as for the above APIs.             */
/*****************************************************************/


/* entrypos is the offset to the start of the list produced by   */
/* QHFLSTFS.                                                     */
  DCL entrypos FIXED BIN(31);

/* numentries is the number of file systems returned by QHFLSTFS. */
  DCL numentries FIXED BIN(31);

/* entrylen is the size of each entry returned by QHFLSTFS.      */
  DCL entrylen FIXED BIN(31);

/* att is used to work with the information in a directory entry. */
  DCL att CHARACTER(276);

/* name is the name of the directory to list.                    */
  DCL name CHARACTER(100);

/* attname and attval are the name and value for an attribute.   */
/* attnamelen and attvallen are the respective lengths.          */
  DCL attname CHARACTER(30);
  DCL attval CHARACTER(30);
  DCL attnamelen FIXED BIN(31);
  DCL attvallen FIXED BIN(31);

/* newname is the value of the attribute QNAME.                  */
  DCL newname CHARACTER(30);

/* filesize is the value of the attribute QFILSIZE.              */
  DCL filesize FIXED BIN(31);

/* fileatt is the value of the attribute QFILATTR.               */
  DCL fileatt CHARACTER(10);

/* chartime is the date/time value for QCRTDTTM.                 */
  DCL 1 chartime,
        3 century CHARACTER(1),
        3 year    CHARACTER(2),
        3 month   CHARACTER(2),
        3 day     CHARACTER(2),
        3 hour    CHARACTER(2),
        3 minute  CHARACTER(2),
        3 second  CHARACTER(2);

/* bytes_used is used to find the beginning of attributes in a   */
/* directory entry.                                              */
  DCL bytes_used FIXED BIN(31);

/* Loop control variable                                         */
  DCL i FIXED BIN(15);




  numptr = ADDR(charbin4);            /* Set numptr, Buf_ptr,    */
  Buf_ptr = ADDR(buffer);             /* SelPtr, and Error_Code. */
  Sel_ptr = ADDR(selection);
  Error_Code = 0;
  name = SUBSTR(Dir_Name,1,INDEX(Dir_Name,' ')); /* Set name     */

  /* If name is blank, list the file systems available.         */
  IF name = ' ' THEN
      DO;
      name = 'ROOT';
      PUT SKIP EDIT('Directory listing for path ',name) (a(27),a(53));
      PUT SKIP;
      PUT SKIP;
```

```
    /* Create the user space for QHFLSTFS to use.               */
    size = 1;
    text = 'temporary user space used by program DIR';
    CALL QUSCRTUS('FSLST     QTEMP    ','TEMPSPACE ',size,' ',
                  '*USE      ',text,'*YES      ',Error_Code);

    /* List the file systems into that space.                   */
    CALL QHFLSTFS('FSLST     QTEMP    ','HFSL0100',Error_Code);

    /* Get the starting point for the file system entries.      */
    startpos = 125;
    len = 4;
    CALL QUSRTVUS('FSLST     QTEMP    ',startpos,len,charbin4,
                  Error_Code);
    entrypos = int;

    /* Get the number of entries in the user space.             */
    startpos = 133;
    len = 4;
    CALL QUSRTVUS('FSLST     QTEMP    ',startpos,len,charbin4,
                  Error_Code);
    numentries = int;

    /* Find the length of the entries.                          */
    startpos = 137;
    len = 4;
    CALL QUSRTVUS('FSLST     QTEMP    ',startpos,len,charbin4,
                  Error_Code);
    entrylen = int;

    /* Loop through the entries and get the names of the file   */
    /* systems.                                                 */
    DO i = 1 to numentries;
      startpos = entrypos + 1;
      len = 10;
      CALL QUSRTVUS('FSLST     QTEMP    ',startpos,len,FSname,
                    Error_Code);
      /* List the names into the spooled file.                  */
      PUT SKIP EDIT(FSname,'<DIR>') ( a(20),A(5) );
      entrypos = entrypos + entrylen;
    END;
    END;
/* If the name was not blank, list that directory.             */
ELSE
    DO;
    PUT SKIP EDIT('Directory listing for path ',name) (a(27),a(53));
    PUT SKIP;
    PUT SKIP;
    /* Build the attribute selection table for QHFOPNDR.   */
    selection_Struct.num_att   = 3;
    selection_Struct.offset1   = 16;
    selection_Struct.offset2   = 28;
    selection_Struct.offset3   = 40;
    selection_Struct.att_len1  = 8;
    selection_Struct.att_name1 = 'QFILSIZE';
    selection_Struct.att_len2  = 8;
    selection_Struct.att_name2 = 'QCRTDTTM';
    selection_Struct.att_len3  = 8;
    selection_Struct.att_name3 = 'QFILATTR';
    selectionlen = 52;
    openinfo = '10    ';                  /* Set openinfo.       */
    namelen = INDEX(NAME,' ') - 1;        /* Set namelen.        */
    /* Open the directory.                                      */
    Call QHFOPNDR(Dir_handle,name,namelen,openinfo,selection,
                  selectionlen,Error_Code);

    /* Read one entry from the directory.                       */
    Call QHFRDDR(Dir_handle,Buffer,300,1,result_count,
                 bytes_returned,Error_Code);

    DO WHILE (result_count > 0);
       attname = '                              ';
       attval  = '                              ';
       att = buffer_Struct.attinfo; /* Set att to Attinfo.      */
       bytes_used = 20; /* 20 equals the amount of data before  */
                        /* attinfo in buffer_struct.            */
       /* Loop for the number of attributes in the entry.       */
       DO i = 1 TO buffer_Struct.num_att;
         charbin4 = SUBSTR(att,1,4);   /* Get attnamelen from   */
         attnamelen = int;             /* the attribute entry.  */
         att = SUBSTR(att,5);          /* Update att.           */
         bytes_used = bytes_used + 4;  /* Update bytes_used.    */
         charbin4 = SUBSTR(att,1,4);   /* Get attvallen from    */
         attvallen = int;              /* the attribute entry.  */
         att = SUBSTR(att,9);          /* Update att.           */
         bytes_used = bytes_used + 8;  /* Update bytes_used.    */
         attname = SUBSTR(att,1,attnamelen);  /* Get attname.   */
         att = SUBSTR(att,attnamelen + 1);    /* Update att.    */
         bytes_used = bytes_used + attnamelen;/* Update bytes_used.*/
         attval = SUBSTR(att,1,attvallen);    /* Get attval.    */
         att = SUBSTR(att,attvallen + 1);     /* Update att.    */
         bytes_used = bytes_used + attvallen; /* Update bytes_used.*/
         /* Update att so that its first character is the first */
         /* character of the next attribute entry.              */
          IF (bytes_used := buffer_struct.offsets(i+1)) &
             ( i = buffer_Struct.num_att) THEN
             att = SUBSTR(att,buffer_struct.offsets(i)-bytes_used+1);

          /* If the attribute is QNAME, then set newname.     */
          IF attname = 'QNAME' THEN
             DO;
             newname = attval;
             END;

          /* If the attribute is QFILSIZE, then set filesize. */
          ELSE IF attname = 'QFILSIZE' THEN
             DO;
             charbin4 = SUBSTR(attval,1,4);
             filesize = int;
             END;

          /* If it was QCRTDTTM, then set time.               */
          ELSE IF attname = 'QCRTDTTM' THEN
             chartime = SUBSTR(attval,1,13);
          /* Else the attribute was QFILATTR, so set fileatt. */
          ELSE
             fileatt = attval;
       END; /* DO */

       /* If the entry was a directory, list its name and     */
       /* and <DIR>.                                          */
       IF (SUBSTR(fileatt,4,1) = '1') THEN
          DO;
          PUT SKIP;
          PUT EDIT(newname,'<DIR>') (a(12),x(3),a(5));
          END;

       /* If the entry is not a hidden file, list its name    */
       /* and size.                                           */
       ELSE IF (SUBSTR(fileatt,2,1) = '1') THEN
          DO;
          PUT SKIP;
          PUT EDIT(newname,filesize) (a(12),f(8));
          END;

       /* If the entry is not a hidden file or directory, list */
       /* its date of creation.                               */
       IF (SUBSTR(fileatt,2,1) = '1') THEN
          DO;
            PUT EDIT(month,'-',day,'-',year)
                   (x(4),a(2),a(1),a(2),a(1),a(2));
            PUT EDIT(hour,':',minute,':',second)
                   (x(3),A(2),a(1),a(2),a(1),a(2));
          END;
       Call QHFRDDR(Dir_handle,Buffer,200,1,result_count,
                    bytes_returned,Error_Code);
    END; /* while */
    END; /* ELSE DO */
/* Close the directory. */
CALL QHFCLODR(Dir_handle,Error_Code);
END DIR;
```

Listing Subdirectories

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  List the subdirectories of the path passed to the      */
/*            program to a spooled file.                             */
/*                                                                   */
```

Examples: Listing Subdirectories

```
/* LANGUAGE:  PL/I                                                  */
/*                                                                  */
/* APIs USED: QHFOPNDR, QHFCLODR, QHFRDDR                           */
/*                                                                  */
/******************************************************************/
/******************************************************************/

DSPSUBDR: PROCEDURE(Dir_Name) OPTIONS(MAIN);


/* Parameter declaration                                            */
DCL Dir_Name CHARACTER (100);

/* API entry declaration                                            */
/*                                                                  */
/* Note that the last parameter, the error code, is declared as     */
/* FIXED BIN(31) and is always zero in the calls to the APIs.       */
/* This specifies that the bytes available are zero and             */
/* exceptions should be signaled.                                   */

DCL QHFOPNDR ENTRY(CHAR(16),CHAR(100),FIXED BIN(31),CHAR(6),
                   CHAR(20),FIXED BIN(31),FIXED BIN(31))
                   OPTIONS(ASSEMBLER);

DCL QHFCLODR ENTRY(CHAR(16),FIXED BIN(31)) OPTIONS(ASSEMBLER);

DCL QHFRDDR ENTRY(CHAR(16),CHAR(200),FIXED BIN(31),FIXED BIN(31),
                  FIXED BIN(31),FIXED BIN(31),FIXED BIN(31))
                  OPTIONS(ASSEMBLER);


ON ERROR SYSTEM;

/* Take only the part of the parameter up until the first blank, */
/* then print the heading to a spooled file and call the         */
/* the procedure to print the rest of the subdirectories.        */

Dir_Name = SUBSTR(Dir_Name,1,INDEX(Dir_Name,' '));
PUT SKIP EDIT('directory substructure starting at ',Dir_Name)
            (A(35),a(45));
CALL Print_Subdir(Dir_Name,0);

/******************************************************************/
/*  Print_SubDir                                                  */
/******************************************************************/
/* This procedure prints out the subdirectory name and reads all */
/* directory entries in name, calling itself recursively         */
/* for any entry that is itself a subdirectory.                  */
/******************************************************************/
/******************************************************************/
Print_SubDir: PROCEDURE (name,numtabs);

  /* Parameter declarations                                         */
  DCL name CHARACTER (100);
  DCL numtabs FIXED BIN(15);

  /****************************************************************/
  /* Parameters for QHFOPNDR                                      */
  /****************************************************************/
  /* Directory handle                                             */
  DCL Dir_Handle CHAR(16);
  /* The paramter name passed to Print_SubDir is used for the     */
  /* path name.                                                   */
  /*                                                              */
  /* Length of directory name                                     */
  DCL namelen FIXED BIN(31);
  /* Open information                                             */
  DCL openinfo CHAR(6);
  /* Attribute selection table                                    */
  DCL selection CHARACTER(20);              /* selection is used  */
  DCL 1 selection_Struct BASED(Sel_ptr),    /* to allocate storage */
          2 num_att FIXED BIN(31),          /* for selection_Struct*/
          2 offset  FIXED BIN(31),          /* this is done by    */
          2 att_len FIXED BIN(31),          /* setting Sel_ptr to */
          2 att_name CHAR(8);               /* the address of     */
  DCL Sel_ptr POINTER;                      /* selection          */
  /* Length of attribute selection table                          */
  DCL selectionlen FIXED BIN(31);
  /* Error code                                                   */
  DCL Error_Code FIXED BIN(31);
  /****************************************************************/

  /****************************************************************/
  /* Parameters for QHFRDDR                                       */
  /****************************************************************/
  /* The directory handle is the same as that for QHFOPNDR.       */
  /*                                                              */
  /* Data buffer                                                  */
  DCL buffer CHARACTER(200);           /* buffer allocates storage */
  DCL 1 buffer_Struct BASED(Buf_ptr),  /* for buffer_Struct        */
          2 num_dir_ret FIXED BIN(31), /* by setting Buf_ptr to    */
          2 offset_dir  FIXED BIN(31), /* the address of buffer.   */
          2 num_att     FIXED BIN(31),
          2 offsets(2)  FIXED BIN(31),
          2 attinfo     CHARACTER(180);
  DCL Buf_ptr POINTER;
  /* Length of data buffer and number of entries to retrieve are  */
  /* hard coded into the API call.                                */
  /*                                                              */
  /* Number of directory entries retrieved                        */
  DCL result_count FIXED BIN(31);
  /* Length of data returned                                      */
  DCL bytes_returned FIXED BIN(31);
  /* The error code is the same as is used for QHFOPNDR.          */
  /****************************************************************/

  /****************************************************************/
  /* Parameters for QHFCLODR.                                     */
  /****************************************************************/
  /* Both the directory handle and the error code are the same as */
  /* those used for QHFOPNDR and QHFRDDR.                         */
  /****************************************************************/

  /* charbin4, int, and numptr are used in conjunction to retrieve */
  /* binary nubers from the datat buffer.                         */
  /* This is done by setting numptr to the address of charbin4.   */
  /* Then, when four bytes are put into charbin4 using SUBSTR, it */
  /* has the effect of reading a binary integer from the buffer.  */
  DCL charbin4 CHARACTER(4);
  DCL int FIXED BIN(31) BASED(numptr);
  DCL numptr POINTER;

  /* bytes_used is used to keep track of the number of bytes used */
  /* from the data buffer.  This is used to get to the            */
  /* beginning of each attribute.                                 */
  DCL bytes_used FIXED BIN(31);

  /* The name and length of the name of an attribute              */
  DCL attname CHARACTER(30);
  DCL attnamelen FIXED BIN(31);

  /* The value and length of the value of an attribute            */
  DCL attvallen FIXED BIN(31);
  DCL attval CHARACTER(30);

  /* The name and length of the name for the attribute QNAME      */
  DCL newname CHARACTER(30);
  DCL newnamelen FIXED BIN(31);

  /* The value of the attribute QFILATTR                          */
  DCL fileatt CHARACTER(9);

  /* Used to manipulate the information in the data buffer        */
  DCL att CHARACTER(180);

  /* tab is set to blanks and used to tab subdirectory entries.   */
  DCL tab CHAR(5);

  /* Loop control variables                                       */
  DCL i FIXED BIN(15);
  DCL j FIXED BIN(15);

/* Start of code for Print_SubDir                                 */
Sel_ptr = ADDR(selection);       /* Set ptr for selection table.  */
Buf_ptr = ADDR(buffer);          /* Set ptr for buffer.           */
openinfo = '10    ';             /* Set open information.         */
selection_Struct.num_att = 1;    /* Constuct the selection table. */
selection_Struct.offset  = 8;
selection_struct.att_len = 8;
selection_struct.att_name = 'QFILATTR';
selectionlen = 20;               /* Set the length of the table.  */
  tab = '      ';                /* tab = 5 spaces.               */
  numptr = ADDR(charbin4);       /* Set address for int.          */
  namelen = INDEX(name,' ') - 1; /* Set namelen.                  */
  Error_Code = 0;                /* Set error code for exceptions.*/
```

```
/* Open the directory.                                          */
Call QHFOPNDR(Dir_handle,name,namelen,openinfo,selection,
              selectionlen,Error_Code);

/* Read one directory entry.                                    */
Call QHFRDDR(Dir_handle,Buffer,200,1,result_count,
             bytes_returned,Error_Code);

PUT SKIP;                       /* Move to next line.           */
DO j = 1 TO NUMTABS;            /* List numtabs tabs.           */
   PUT EDIT(tab) (a(5));
END;
PUT EDIT(name) (a(30));         /* Write directory name         */
                                /* to splooled file.            */

DO WHILE (result_count > 0);/* While read directory was successful*/
   attname = '                          ';
   attval  = '                          ';
   att = buffer_Struct.attinfo; /* Set att to buffer_Struct.attinfo*/
   bytes_used = 12; /* Number of bytes in att table before attinfo */
   DO i = 1 TO buffer_Struct.num_att;
      charbin4 = SUBSTR(att,1,4); /* Read attribute name length    */
      attnamelen = int;           /* from att and set attnamelen.  */
      att = SUBSTR(att,5);        /* Update att and bytes_used.    */
      bytes_used = bytes_used + 4;
      charbin4 = SUBSTR(att,1,4); /* Read attribute value length   */
      attvallen = int;            /* from att and set attvallen.   */
      att = SUBSTR(att,9);        /* Update att and bytes_used.    */
      bytes_used = bytes_used + 8;
      attname = SUBSTR(att,1,attnamelen);   /* Set attname.        */
      att = SUBSTR(att,attnamelen + 1);     /* Update att.         */
      bytes_used = bytes_used + attnamelen; /* Update bytes_used.  */
      attval = SUBSTR(att,1,attvallen);     /* Set attval.         */
      att = SUBSTR(att,attvallen + 1);      /* Update att and      */
      bytes_used = bytes_used + attvallen;  /* bytes_used.         */

   /* Set att to next attribute using bytes_used and the next      */
   /* attribute's offset.                                          */
      IF (bytes_used := buffer_struct.offsets(i+1)) &
         ( i := buffer_Struct.num_att) THEN
         att = SUBSTR(att,buffer_struct.offsets(i) - bytes_used + 1);

   /* If attribute is QNAME, set newname and newnamelen            */
   /* in case the entry is a directory.                            */
      IF attname = 'QNAME' THEN
         DO;
         newname = attval;
         newnamelen = attvallen;
         END;
   /* Else attribute was QFILATTR, so set fileatt.                 */
      ELSE
         fileatt = attval;
   END; /* DO */

   /* If the entry was a directory                                 */
   IF (SUBSTR(fileatt,4,1) = '1') THEN
      DO;
      /* Construct path name and call Print_SubDir to print        */
      /* its subdirectories.                                       */
      newname = SUBSTR(name,1,namelen)||'/'||
                SUBSTR(newname,1,newnamelen);
      CALL Print_Subdir(newname,numtabs + 1);
      END;
   /* Read next directory entry                                    */
   Call QHFRDDR(Dir_handle,Buffer,200,1,result_count,
                bytes_returned,Error_Code);
  END; /* while */
  /* Close the directory                                          */
  CALL QHFCLODR(Dir_Handle,Error_Code);
END Print_Subdir;


END DSPSUBDR;
```

Working with Stream Files

The following C/400 program performs the following functions:

- Opens an existing stream file
- Creates or replaces a database file
- Reads from the stream file and writes to the database file until end-of-file
- Closes both files

The program uses the following hierarchical file system (HFS) APIs:

- Open Stream File (QHFOPNSF)
- Read from Stream File (QHFRDSF)
- Close Stream File (QHFCLOSF)

```
/*********************************************************************/
/* Program Name:  HFSCOPY  C                                         */
/* Language    :  C/400                                              */
/* Description :  This program will do the following:                */
/*                -- Create or replace a stream file                 */
/*                -- Create or replace a database file             */
/*                -- Read from the stream file and write to the      */
/*                   database file until EOF                         */
/*                -- Close both files when done                      */
/*********************************************************************/

/*********************************************************************/
/* Include files                                                     */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ophfapi.h>


/*********************************************************************/
/* Structure and variable definitions                                */
/*********************************************************************/

#define ON   1
#define OFF  0
typedef struct error_code_struct {
               int  bytes_provided;
               int  bytes_available;
               char exception_id[7];
               char reserved_field;
               char exception_data[256];
              }error_code_struct;
error_code_struct error_code;
handle_struct file_handle;
char path_name[30];
char open_info[10];
char attrib_info;
char action;
char read_buffer[80];
int  path_length;
int  attrib_length = 0;
int  bytes_to_read;
int  bytes_read = 0;
int end_file;
int cmpgood;
FILE *FP;

/*********************************************************************/
/*printErrCode: Routine to print the error code structure            */
/*********************************************************************/
void printErrCode(error_code_struct *theErrCode)
{
   int i;
   char *tempptr = theErrCode->exception_id;
   printf("Bytes Provided -> %d\n",theErrCode->bytes_provided);
   printf("Bytes Available -> %d\n",theErrCode->bytes_available);
   printf("Exception ID -> ");
   for (i=0;i<7 ;i++,tempptr++ )
     {
       putchar(*tempptr);
```

```
    }
  putchar('\n');
}


/*******************************************************************/
/* Start of code                                                   */
/*******************************************************************/
main()
{

 error_code.bytes_provided = 116;
 strcpy(path_name,"/QDLS/HFSFLR/SAMPLE.HFS");
 path_length = strlen(path_name);

/*******************************************************************/
/* Open the stream file                                            */
/*******************************************************************/
 strcpy(open_info,"210 120   ");    /* Create or replace the file   */
 printf("OPEN STREAM FILE:\n  ");
 QHFOPNSF(&file_handle,
          path_name,
          path_length,
          open_info,
          &attrib_info,
          attrib_length,
          &action,
          &error_code);
 if (error_code.bytes_available := 0)
   {
    printErrCode(&error_code);
    exit(1);
   }

/*******************************************************************/
/* Open a database file                                            */
/*******************************************************************/

 if (( FP = fopen("HFSLIB/HFSFILE(SAMPLE)","wb")) == NULL)
   {
    printf("Cannot open HFSLIB/HFSFILE(SAMPLE)\n");
    exit(1);
   }

/*******************************************************************/
/* Loop through reading from the stream file and writing to the    */
/* database file.                                                  */
/*******************************************************************/

 end_file = OFF;
 while (end_file == OFF)
   {
     /*************************************************************/
     /* Read 80 bytes from the stream file                        */
     /*************************************************************/
     bytes_to_read = 80;
     printf("READ STREAM FILE:\n  ");
     QHFRDSF(&file_handle,
             read_buffer,
             bytes_to_read,
             &bytes_read,
             &error_code);
     if (error_code.bytes_available != 0)
       {
         cmpgood = strncmp("CPF1F33",error_code.exception_id,7);
         if (cmpgood != 0)
           printErrCode(&error_code);
         end_file = ON;
       }
     else
       {
         printf("BYTES READ: %d\n  ",bytes_read);
         printf("READ BUFFER: %s\n",read_buffer);
         if (bytes_read < bytes_to_read)
           {
             end_file = ON;
             /*****************************************************/
             /* Write remaining bytes to the database file        */
             /*****************************************************/
             if (bytes_read > 0)
               fwrite(read_buffer,1,bytes_read,FP);
           }
       }
   }

/*******************************************************************/
/* Close the stream file                                           */
/*******************************************************************/
 printf("CLOSE STREAM FILE:\n  ");
 QHFCLOSF(&file_handle,
          &error_code);
 if (error_code.bytes_available != 0)
   printErrCode(&error_code);

/*******************************************************************/
/* Close the database file                                         */
/*******************************************************************/
 fclose(FP);
}
```

Using SNA Management Services Transport APIs

This example shows a source and target application using network management transport APIs to send and receive management services data.

Source Application Program

This source application program sends a request to a target application.

```
/*-----------------------------------------------------------------*/
/* This is a source application that uses the management services  */
/* transport APIs.  It does the following:                         */
/*   1.  Prompts for the network ID and CP name of the remote system */
/*       where target application MSTTARG has been started.        */
/*   2.  Prompts for data to be sent to MSTTARG.                   */
/*   3.  Prompts for whether or not a reply is required.           */
/*   4.  Sends a management services transport request to MSTTARG. */
/*   5.  Repeats steps 2-4 until QUIT is entered.                  */
/*                                                                 */
/* Note: MSTTARG may be ended by this application by sending it the */
/*       the string "ENDRMTAPP".                                   */
/*-----------------------------------------------------------------*/
/*-----------------------------------------------------------------*/
/* Includes                                                        */
/*-----------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define NOERROR "NOERROR"
#define RQSONLY "*RQS      "
#define RQSRPY "*RQSRPY   "

/*-----------------------------------------------------------------*/
/* Type definitions                                                */
/*-----------------------------------------------------------------*/
typedef int HANDLE;                /* typedef for handle           */
typedef char APPLNAME[8];          /* typedef for application name */
typedef char NETID[8];             /* typedef for network ID       */
typedef char CPNAME[8];            /* typedef for control point name*/
typedef char MODENAME[8];          /* typedef for mode name        */
typedef char SENSECODE[8];         /* typedef for SNA sense code (in
                                      character format)            */
typedef char LIBNAME[10];          /* typedef for library name     */
typedef char QNAME[10];            /* typedef for data queue name  */
typedef char MSGID[7];             /* typedef for message ID       */
typedef char EXCPDATA[48];         /* typedef for exception data   */
typedef char CATEGORY[8];          /* typedef for category         */
typedef char APPLTYPE[10];         /* typedef for application type */
typedef char REPLREG[10];          /* typedef for replace
                                      registration                 */
typedef char DATARCVD[10];         /* typedef for data received    */
typedef char REQTYPE[10];          /* typedef for request type     */
typedef char POSTRPL[10];          /* typedef for post reply       */
typedef char REQUESTID[53];        /* typedef for request ID       */
typedef char SRBUFFER[500];        /* typedef for send/receive
                                      buffer.  This program limits
                                      the amount of data to be sent
```

```
                                    or received to 500 bytes.  The
                                    maximum size of a management
                                    services transport buffer is
                                    31739.                          */

typedef struct {                    /* Library-qualified data queue
                                    name                            */
   QNAME data_queue_name;           /* data queue name                 */
   LIBNAME library_name;            /* library name                    */
  } QUALQNAME;

typedef struct {                    /* Error code structure            */
   int bytes_provided;              /* number of bytes provided        */
   int bytes_available;             /* number of bytes available       */
   MSGID exception_ID;              /* exception ID                    */
   char reserved_area;              /* reserved                        */
   EXCPDATA exception_data;         /* exception data                  */
  } ERRORCODE;

typedef struct {                    /* Notification record structure */
   char record_type[10];            /* Record type                     */
   char function[2];                /* Function                        */
   HANDLE handle;                   /* Handle                          */
   REQUESTID req_id;                /* Request ID                      */
   char reserved[11];               /* Reserved area                   */
  } NOTIFRCD;

typedef struct {                    /* Receiver variable structure   */
   int bytes_provided;              /* number of bytes provided        */
   int bytes_available;             /* number of bytes available       */
   SRBUFFER received_data;          /* received data                   */
  } RECEIVERVAR;

typedef struct {                    /* Qualified application name    */
   NETID network_id;                /* Network ID                      */
   CPNAME cp_name;                  /* Control point name              */
   APPLNAME app_name;               /* Application name                */
  } QUALAPPL;

/*-------------------------------------------------------------------*/
/* External program declarations                                     */
/*-------------------------------------------------------------------*/
#pragma linkage(QNMSTRAP, OS)       /* Start application API         */
extern void QNMSTRAP (HANDLE *handle, /* pointer to handle           */
                      APPLNAME *applname, /* pointer to appl name    */
                      QUALQNAME *qualqname, /* pointer to data queue
                                    name                            */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

#pragma linkage(QNMENDAP, OS)       /* End application API           */
extern void QNMENDAP (HANDLE *handle, /* pointer to handle           */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

#pragma linkage(QNMRCVDT, OS)       /* Receive data API              */
extern void QNMRCVDT (HANDLE *handle, /* pointer to handle           */
                      RECEIVERVAR *rcvvar, /* pointer to receiver
                                    variable                        */
                      int *rcvvarln, /* pointer to receiver variable
                                    length                          */
                      REQUESTID *reqid, /* pointer to request ID     */
                      QUALAPPL *qualappl, /* pointer to remote
                                    application name                */
                      DATARCVD *datarcvd, /* pointer to type of data
                                    received                        */
                      int *waittim, /* pointer to wait time          */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

#pragma linkage(QNMSNDRQ, OS)       /* Send request API              */
extern void QNMSNDRQ (HANDLE *handle, /* pointer to handle           */
                      QUALAPPL *qualappl, /* pointer to remote
                                    application name                */
                      REQUESTID *reqid, /* pointer to request ID     */
                      SRBUFFER *sndbuf, /* pointer to send buffer    */
                      int *sndbufln, /* pointer to send buffer length*/
                      REQTYPE *reqtype, /* pointer to request type   */
                      POSTRPL *postrpl, /* pointer to post reply     */
                      int *waittim, /* pointer to wait time          */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

#pragma linkage(QNMCHGMN, OS)       /* Change mode name API          */
extern void QNMCHGMN (HANDLE *handle, /* pointer to handle           */
                      MODENAME *modename, /* pointer to mode name    */
                      ERRORCODE *errorcode); /* pointer to error code
                                    parameter                       */

void check_error_code (char func_name[8]); /* Used to check error code
                                                                    */
void get_network_id (void);         /* Get network ID of destination
                                    node                            */
void get_cp_name (void);            /* Get CP name of destination
                                    node                            */
void process_replies(void);         /* Process replies received from
                                    destination application         */

/*-------------------------------------------------------------------*/
/* Global declarations                                               */
/*-------------------------------------------------------------------*/
 HANDLE appl_handle;                /* Handle of application         */
 ERRORCODE error_code_struc =       /* Error code parameter          */
          {sizeof(error_code_struc), /* Initialize bytes provided  */
           0,                       /* initialize bytes available    */
           NOERROR};                /* initialize error code         */
 char input_line[80];               /* Input data                    */
 QUALAPPL qual_appl =               /* Qualified application name    */
       {"        ","        ","        "};
 REQUESTID req_id;                  /* Returned request ID           */
 int wait_time = -1;                /* Wait time = wait forever      */

/*-------------------------------------------------------------------*/
/* Start of main.                                                    */
/*-------------------------------------------------------------------*/
int main ()
{
 APPLNAME appl_name = "MSTSOURC";   /* Application name to be used   */
 QUALQNAME data_queue_parm =        /* Data queue name to be used    */
          {"*NONE     ", "         "}; /* Initialize structure     */
 NOTIFRCD notif_record;             /* Area to contain notification
                                    record                          */
 CATEGORY category = "*NONE   ";    /* SNA/Management Services function
                                    set group                       */
 APPLTYPE appl_type = "*FPAPP    "; /* Application type              */
 REPLREG replace_reg = "*YES      "; /* Replace registration = *YES  */
 int sys_result;                    /* Result of system function     */
 char end_msg[] = "ENDRMTAPPL";     /* If this data is received then
                                    the application will end        */
 char incoming_data[] = "01";       /* Incoming data constant        */
 SRBUFFER send_buffer;              /* Send buffer                   */
 int data_length;                   /* Length of send data           */
 char input_char;                   /* Input character               */
 REQTYPE req_type;                  /* Request type                  */
 POSTRPL post_reply = "*NO       "; /* Don't post any received replies
                                                                    */
 MODENAME mode_name = "#INTER  ";   /* Mode name = #INTER            */

/*-------------------------------------------------------------------*/
/* Start of code                                                     */
/*-------------------------------------------------------------------*/
 QNMSTRAP (&appl_handle,
           &appl_name,
           &data_queue_parm,
           &error_code_struc);      /* Start application             */
 check_error_code("QNMSTRAP");      /* Check error code              */
 QNMCHGMN (&appl_handle,
           &mode_name,
           &error_code_struc);      /* Change mode name              */
 check_error_code("QNMCHGMN");      /* Check error code              */
 get_network_id();                  /* Get network ID                */
 get_cp_name();                     /* Get CP name                   */
 memcpy(qual_appl.app_name,
        "MSTTARG ",
        sizeof(qual_appl.app_name)); /* Copy application name        */
 printf ("Enter message to send to remote application or "
         "QUIT to end\n");
 gets(input_line);
 while (memcmp(input_line,
               "QUIT",
               sizeof("QUIT")) != 0) /* While an ending string
                                    has not been entered            */
   {
     data_length = strlen(input_line); /* Get length of message     */
     memcpy(send_buffer,
            input_line,
```

```
          data_length);          /* Put message in send buffer    */
    printf("Reply necessary? (Y or N)\n"); /* Prompt for reply
                                     indicator                     */
    gets(input_line);            /* Get reply character           */
    input_char = toupper(input_line[0]); /* Convert character to
                                     uppercase                     */
    while (strlen(input_line) != 1 ||
           (input_char != 'Y' &&
           input_char != 'N'))
      {
        printf("Please type Y or N\n");
        gets(input_line);        /* Get reply character           */
        input_char = toupper(input_line[0]); /* Convert character to
                                     uppercase                     */
      }
    if (input_char == 'Y')
      {
        memcpy(req_type,
               RQSRPY,
               sizeof(req_type)); /* Indicate request should have
                                     a reply                       */
      }
    else
      {
        memcpy(req_type,
               RQSONLY,
               sizeof(req_type)); /* Indicate request should not have
                                     a reply                       */
      }
    QNMSNDRQ (&appl_handle,
              &qual_appl,
              &req_id,
              &send_buffer,
              &data_length,
              &req_type,
              &post_reply,
              &wait_time,
              &error_code_struc); /* Send request to remote
                                     application                   */
    check_error_code("QNMSNDRQ"); /* Check error code              */
    if (input_char == 'Y')
      {
        process_replies();       /* Process one or more received
                                     replies                       */
      }
    printf ("Enter message to send to remote application or "
            "QUIT to end\n");
    gets(input_line);
  }
 QNMENDAP (&appl_handle,
           &error_code_struc);   /* End the application           */

 return 0;
}

/*--------------------------------------------------------------------*/
/* process_replies function                                           */
/*--------------------------------------------------------------------*/
void process_replies ()
{
 RECEIVERVAR receiver_var =      /* Receiver variable             */
           {sizeof(receiver_var)}; /* Initialize bytes provided   */
 int rcv_var_len = sizeof(receiver_var); /* Length of receiver
                                     variable                      */
 DATARCVD data_rcvd = "*NODATA   "; /* Type of data received       */
 QUALAPPL qual_appl;             /* Sender of reply               */

 printf ("Received reply(s):\n");
 while (memcmp(data_rcvd,
               "*RPYCPL   ",
               sizeof(data_rcvd)) != 0) /* While final reply has not
                                     been received                 */
   {
     strncpy(receiver_var.received_data,
             "\0",
             sizeof(receiver_var.received_data)); /* Null out
                                     data buffer                   */
     QNMRCVDT (&appl_handle,
               &receiver_var,
               &rcv_var_len,
               &req_id,
               &qual_appl,
               &data_rcvd,
               &wait_time,
               &error_code_struc); /* Receive reply               */
     check_error_code("QNMRCVDT"); /* Check error code            */
     printf("%1.500s\n",receiver_var.received_data); /* Print out
                                                        reply      */
   }
}

/*--------------------------------------------------------------------*/
/* get_network_id function.                                           */
/*--------------------------------------------------------------------*/
void get_network_id ()
{
 int count;
 printf("Enter network ID of remote system where MSTTARG "
        "application has been started\n"); /* Prompt for network
                                     ID                            */
 gets(input_line);               /* Get network ID                */
 while (strlen(input_line) <= 0 ||
        strlen(input_line) > 8)  /* While network ID is not valid */
   {
     printf("Network ID is too long or too short - try again\n");
     gets(input_line);           /* Get network ID                */
   }
 memcpy(qual_appl.network_id,
        input_line,
        strlen(input_line));     /* Copy network ID               */
 for (count=0; count < strlen(input_line); count++)
   qual_appl.network_id[count] =
                 toupper(qual_appl.network_id[count]); /* Convert
                                     input to uppercase            */
}

/*--------------------------------------------------------------------*/
/* get_cp_name function.                                              */
/*--------------------------------------------------------------------*/
void get_cp_name ()
{
 int count;
 printf("Enter CP name of remote system where MSTTARG application "
        "has been started\n");   /* Prompt for CP name            */
 gets(input_line);               /* Get CP name                   */
 while (strlen(input_line) <= 0 ||
        strlen(input_line) > 8)  /* While CP name is not valid    */
   {
     printf("CP name is too long or too short - try again\n");
     gets(input_line);           /* Get CP name                   */
   }
 memcpy(qual_appl.cp_name,
        input_line,
        strlen(input_line));     /* Copy CP name                  */
 for (count=0; count < strlen(input_line); count++)
   qual_appl.cp_name[count] =
                   toupper(qual_appl.cp_name[count]); /* Convert
                                     input to uppercase            */
}

/*--------------------------------------------------------------------*/
/* check_error_code -                                                 */
/*--------------------------------------------------------------------*/
void check_error_code (char func_name[8])
{
 char *sense_ptr = error_code_struc.exception_data + 36; /*
                                     Pointer to sense code in
                                     exception data                */
 SENSECODE sense_code;           /* SNA sense code                */
 if (error_code_struc.bytes_available != 0) /* Error occurred?    */
   {
     printf("\n\nError occurred calling %1.8s.\n",func_name);
     memcpy(sense_code,
            sense_ptr,
            sizeof(sense_code));  /* Copy sense code from exception
                                     data                          */
     printf("Error code is %1.7s, SNA sense code is %1.8s.\n",
                 error_code_struc.exception_ID,
                 sense_code);
     if (memcmp(func_name,
            "QNMSTRAP",
            sizeof(func_name)) != 0)  /* Error did not occur on
                                     start application?            */
       {
         QNMENDAP (&appl_handle,
                   &error_code_struc); /* End the application     */
```

```
      }
     exit(EXIT_FAILURE);           /* Exit this program            */
   }
}
```

Target Application Program

This target application receives requests from and returns replies to source applications.

```
/*--------------------------------------------------------------------*/
/* This is a target application that uses the management services    */
/* transport APIs.  It receives management services transport       */
/* requests from source application MSTSOURC and displays the data   */
/* contained in the request.  If the request specifies that a       */
/* reply needs to be sent, this program accepts input from the      */
/* keyboard and sends one or more replies to the source application. */
/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
/* Includes                                                           */
/*--------------------------------------------------------------------*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define NOERROR "NOERROR"
#define REQUEST "*RQS      "
#define REQREPLY "*RQSRPY   "
#define REPLYINC "*RPYINCPL "
#define REPLYCMP "*RPYCPL   "

/*--------------------------------------------------------------------*/
/* Type definitions                                                   */
/*--------------------------------------------------------------------*/
typedef int HANDLE;                /* typedef for handle           */
typedef char APPLNAME[8];          /* typedef for application name  */
typedef char NETID[8];             /* typedef for network ID        */
typedef char CPNAME[8];            /* typedef for control point name*/
typedef char SENSECODE[8];         /* typedef for SNA sense code
                                      (in character format)        */

typedef char LIBNAME[10];          /* typedef for library name      */
typedef char QNAME[10];            /* typedef for data queue name   */
typedef char MSGID[7];             /* typedef for message ID        */
typedef char EXCPDATA[48];         /* typedef for exception data    */
typedef char CATEGORY[8];          /* typedef for category          */
typedef char APPLTYPE[10];         /* typedef for application type  */
typedef char REPLREG[10];          /* typedef for replace
                                      registration                 */
typedef char DATARCVD[10];         /* typedef for data received     */
typedef char REPLYTYPE[10];        /* typedef for reply type        */
typedef char REQUESTID[53];        /* typedef for request ID        */
typedef char PACKED5[3];           /* typedef for PACKED(5,0) field */
typedef char SRBUFFER[500];        /* typedef for send/receive
                                      buffer.  This program limits
                                      the amount of data to be sent
                                      or received to 500 bytes.  The
                                      maximum size of a management
                                      services transport buffer is
                                      31739.                       */

typedef struct {                   /* Library-qualified data queue
                                      name                         */
   QNAME data_queue_name;          /* data queue name              */
   LIBNAME library_name;           /* library name                 */
  } QUALQNAME;

typedef struct {                   /* Error code structure         */
   int bytes_provided;             /* number of bytes provided     */
   int bytes_available;            /* number of bytes available    */
   MSGID exception_ID;             /* exception ID                 */
   char reserved_area;             /* reserved                     */
   EXCPDATA exception_data;        /* exception data               */
  } ERRORCODE;

typedef struct {                   /* Notification record structure */
   char record_type[10];           /* Record type                  */
   char function[2];               /* Function                     */
   HANDLE handle;                  /* Handle                       */
   REQUESTID req_id;               /* Request ID                   */
   char reserved[11];              /* Reserved area                */
  } NOTIFRCD;

typedef struct {                   /* Receiver variable structure  */
   int bytes_provided;             /* number of bytes provided     */
   int bytes_available;            /* number of bytes available    */
   SRBUFFER received_data;         /* received data                */
  } RECEIVERVAR;

typedef struct {                   /* Qualified application name   */
   NETID network_id;               /* Network ID                   */
   CPNAME cp_name;                 /* Control point name           */
   APPLNAME app_name;              /* Application name             */
  } QUALAPPL;

/*--------------------------------------------------------------------*/
/* External program declarations                                      */
/*--------------------------------------------------------------------*/
#pragma linkage(QNMSTRAP, OS)      /* Start application API        */
extern void QNMSTRAP (HANDLE *handle, /* pointer to handle         */
                      APPLNAME *applname,  /* pointer to application
                                      name                         */
                      QUALQNAME *qualqname,/* pointer to data queue
                                      name                         */
                      ERRORCODE *errorcode); /* pointer to error code
                                      parameter                    */

#pragma linkage(QNMENDAP, OS)      /* End application API          */
extern void QNMENDAP (HANDLE *handle, /* pointer to handle         */
                      ERRORCODE *errorcode); /* pointer to error code
                                      parameter                    */

#pragma linkage(QNMREGAP, OS)      /* Register application API     */
extern void QNMREGAP (HANDLE *handle, /* pointer to handle         */
                      CATEGORY *category, /* pointer to category    */
                      APPLTYPE *appltype, /* pointer to application
                                      type                         */
                      REPLREG *replreg, /* pointer to replace
                                      registration parameter       */
                      ERRORCODE *errorcode); /* pointer to error code
                                      parameter                    */

#pragma linkage(QNMDRGAP, OS)      /* Deregister application API   */
extern void QNMDRGAP (HANDLE *handle, /* pointer to handle         */
                      ERRORCODE *errorcode); /* pointer to error code
                                       set group                   */

#pragma linkage(QNMRCVDT, OS)      /* Receive data API             */
extern void QNMRCVDT (HANDLE *handle, /* pointer to handle         */
                      RECEIVERVAR *rcvvar, /* pointer to receiver
                                      variable                     */
                      int *rcvvarln, /* pointer to receiver variable
                                      length                       */
                      REQUESTID *reqid, /* pointer to request ID    */
                      QUALAPPL *qualappl, /* pointer to remote
                                      application name             */
                      DATARCVD *datarcvd, /* pointer to type of data
                                      received                     */
                      int *waittim, /* pointer to wait time         */
                      ERRORCODE *errorcode); /* pointer to error code
                                      parameter                    */

#pragma linkage(QNMSNDRP, OS)      /* Send reply API               */
extern void QNMSNDRP (HANDLE *handle, /* pointer to handle         */
                      REQUESTID *reqid, /* pointer to request ID    */
                      SRBUFFER *sndbuf, /* pointer to send buffer   */
                      int *sndbufln, /* pointer to send buffer length*/
                      REPLYTYPE *rpltype, /* pointer to reply type  */
                      int *waittim, /* pointer to wait time         */
                      ERRORCODE *errorcode); /* pointer to error code
                                      parameter                    */

#pragma linkage(QNMRCVOC, OS)      /* Receive operation completion API
                                                                    */
extern void QNMRCVOC (HANDLE *handle, /* pointer to handle         */
                      REQUESTID *reqid, /* pointer to request ID    */
                      QUALAPPL *qualappl, /* pointer to remote
                                      application name             */
                      ERRORCODE *errorcode); /* pointer to error code
                                      parameter                    */

#pragma linkage(QRCVDTAQ, OS)      /* Receive data queue           */
extern void QRCVDTAQ (QNAME *queue_name, /* pointer to queue name   */
                      LIBNAME *lib_name, /* pointer to library name */
```

```
                    PACKED5 *rcd_len, /* pointer to record length  */
                    NOTIFRCD *notifrcd, /* pointer to notification
                                record                             */
                    PACKED5 *waittime); /* pointer to wait time    */

void check_error_code (char func_name[8]); /* Used to check error
                                code                               */

/*--------------------------------------------------------------------*/
/* Global declarations                                                */
/*--------------------------------------------------------------------*/
 HANDLE appl_handle;               /* Handle of application          */
 ERRORCODE error_code_struc =      /* Error code parameter           */
          {sizeof(error_code_struc), /* Initialize bytes provided   */
           0,                      /* initialize bytes available     */
           NOERROR};               /* initialize error code          */
/*--------------------------------------------------------------------*/
/* Start of main function                                             */
/*--------------------------------------------------------------------*/
int main ()
{
/*--------------------------------------------------------------------*/
/* Local declarations                                                 */
/*--------------------------------------------------------------------*/
 APPLNAME appl_name = "MSTTARG ";  /* Application name to be used    */
 QUALQNAME data_queue_parm =       /* Data queue name to be used     */
          {"MSTDTAQ   ", "QTEMP     "}; /* Initialize structure     */
 NOTIFRCD notif_record;            /* Area to contain notification
                                      record                        */
 RECEIVERVAR receiver_var =        /* Receiver variable              */
          {sizeof(receiver_var)};  /* Initialize bytes provided      */
 QUALAPPL qual_appl;               /* Qualified application name     */
 DATARCVD data_rcvd;               /* Type of data received          */
 CATEGORY category = "*NONE   ";   /* SNA/Management Services function
                                      set group                     */
 APPLTYPE appl_type = "*FPAPP    "; /* Application type              */
 REPLREG replace_reg = "*YES      "; /* Replace registration = *NO   */
 REPLYTYPE reply_cmp = REPLYCMP;   /* Complete reply                 */
 REPLYTYPE reply_inc = REPLYINC;   /* Incomplete reply               */
 int sys_result;                   /* Result of system function      */
 int rcv_var_len = sizeof(receiver_var); /* Length of receiver
                                      variable                      */
 PACKED5 wait_time_p = "\x00\x00\x1D"; /* Packed value for wait time
                                      = -1, i.e. wait forever       */
 PACKED5 record_len;               /* Length of received data queue
                                      record                        */
 int wait_forever = -1;            /* Integer value for wait time =
                                      -1, that is, wait forever     */
 int no_wait = 0;                  /* Do not wait for I/O to
                                      complete                      */
 char end_msg[] = "ENDRMTAPPL";    /* If this data is received then
                                      the application will end      */
 char incoming_data[] = "01";      /* Incoming data constant         */
 char inbuf[85];                   /* Input buffer                   */
 SRBUFFER send_buffer;             /* Send buffer for sending
                                      replies                       */
 int reply_len;                    /* Length of reply data           */

/*--------------------------------------------------------------------*/
/* Start of executable code                                           */
/*--------------------------------------------------------------------*/
 sys_result = system("DLTDTAQ DTAQ(QTEMP/MSTDTAQ)"); /* Delete
                                      previous data queue (if any)  */
 sys_result = system("CRTDTAQ DTAQ(QTEMP/MSTDTAQ) MAXLEN(80)"); /*
                                      Create data queue             */
 QNMSTRAP (&appl_handle,
           &appl_name,
           &data_queue_parm,
           &error_code_struc);     /* Start application              */
 check_error_code("QNMSTRAP");     /* Check error code               */
 QNMREGAP (&appl_handle,
           &category,
           &appl_type,
           &replace_reg,
           &error_code_struc);     /* Register the application       */
 check_error_code("QNMREGAP");     /* Check error code               */
 while (memcmp(receiver_var.received_data,
               end_msg,
               sizeof(end_msg)) != 0)
   {                               /* Loop until an ending string
                                      has been sent by the requesting
                                      application                   */
     QRCVDTAQ (&data_queue_parm.data_queue_name,
               &data_queue_parm.library_name,
               &record_len,
               &notif_record,
               &wait_time_p);      /* Receive indication from data
                                      queue                         */
     if (memcmp(notif_record.function,
                incoming_data,
                sizeof(incoming_data)) == 0) /* Incoming data was
                                      received?                     */
       {
         strncpy(receiver_var.received_data,
                 "\0",
                 sizeof(receiver_var.received_data)); /* Null out the
                                      receive buffer                */
         QNMRCVDT (&appl_handle,
                   &receiver_var,
                   &rcv_var_len,
                   &notif_record.req_id,
                   &qual_appl,
                   &data_rcvd,
                   &wait_forever,
                   &error_code_struc); /* Receive data using the
                                      request ID in the notification*/
         check_error_code("QNMRCVDT"); /* Check error code          */
         printf("%1.500s\n",receiver_var.received_data); /* Display
                                      the received data             */
         if (memcmp(data_rcvd,
                    REQREPLY,
                    sizeof(data_rcvd)) == 0) /* Request requires
                                      a reply?                      */
           {
             printf("Please enter your replies (a null line "
                    "indicates that you are finished)\n"); /* Display
                                      a prompt message              */
             gets(inbuf);          /* Get the reply data             */
             reply_len = strlen(inbuf); /* Get length of reply       */
             while (reply_len != 0)  /* While no null string was input
                                                                    */
               {
                 memcpy(send_buffer,inbuf,strlen(inbuf)); /* Copy
                                      data to send buffer           */
                 QNMSNDRP (&appl_handle,
                           &notif_record.req_id,
                           &send_buffer,
                           &reply_len,
                           &reply_inc,
                           &no_wait,
                           &error_code_struc); /* Send a reply to the
                                      source application (specify
                                      "not last" reply).  The results
                                      of this operation will be
                                      obtained later using the
                                      receive operation completion
                                      API.                          */
                 gets(inbuf);       /* Get the next reply            */
                 reply_len = strlen(inbuf); /* Get length of reply   */
               }
             QNMSNDRP (&appl_handle,
                       &notif_record.req_id,
                       &send_buffer,
                       &reply_len,
                       &reply_cmp,
                       &no_wait,
                       &error_code_struc); /* Send final reply (this
                                      contains no data).  The results
                                      of this operation will be
                                      obtained later using the
                                      receive operation completion
                                      API.                          */
           }
         else
           {                          /* A reply is not required    */
             if (memcmp(data_rcvd,
                 REQUEST,
                 sizeof(data_rcvd)) != 0) /* Something other than a
                                      request was received?         */
               {
                 printf("Incorrect data was received, "
                        "data_rcvd = %1.10s\n", data_rcvd); /* Print
                                      value of data_rcvd            */
               }
           }
```

```
      }
    else
      {                              /* A send completion was received
                                        for a previous send reply
                                        operation                    */
        QNMRCVOC (&appl_handle,
                  &notif_record.req_id,
                  &qual_appl,
                  &error_code_struc);/* Receive operation completion*/
        check_error_code("QNMRCVOC"); /* Check error code           */
        printf("Reply was sent successfully.\n"); /* Error code was
                                        OK                           */
      }
  }
 QNMDRGAP (&appl_handle,
           &error_code_struc);       /* Deregister the application   */
 QNMENDAP (&appl_handle,
           &error_code_struc);       /* End the application          */

 return 0;
}

/*--------------------------------------------------------------------*/
/* check_error_code -                                                 */
/*                                                                    */
/*   This function validates the error code parameter returned on     */
/*   the call to a management services transport API program.  If     */
/*   an error occurred, it displays the error that occurred and       */
/*   ends this program.                                               */
/*--------------------------------------------------------------------*/
void check_error_code (char func_name[8])
{
 char *sense_ptr = error_code_struc.exception_data + 36; /*
                                        Pointer to sense code in
                                        exception data               */
 SENSECODE sense_code;              /* SNA sense code                */
 if (error_code_struc.bytes_available != 0) /* Error occurred?       */
   {
    printf("\nError occurred calling %1.8s.\n",func_name);
    memcpy(sense_code,
           sense_ptr,
           sizeof(sense_code));     /* Copy sense code from exception
                                        data                         */
    printf("Error code is %1.7s, SNA sense code is %1.8s.\n",
                error_code_struc.exception_ID,
                sense_code);
    if (memcmp(func_name,
           "QNMSTRAP",
           sizeof("QNMSTRAP")) != 0)  /* Error did not occur on
                                        start application?           */
      {
        QNMENDAP (&appl_handle,
                  &error_code_struc); /* End the application        */

      }
    exit(EXIT_FAILURE);             /* Exit this program            */
   }
}
```

Using COBOL Program to Call APIs

This example COBOL program uses the example error handler in “Error Handler for Example COBOL Program” on page A-34.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ACF24.
*****************************************************************
*                                                               *
* THE PURPOSE OF THIS PROGRAM IS TO SHOW HOW TO CALL THE        *
* VARIOUS APIs, WHILE TESTING THAT THEY WORK PROPERLY.          *
*                                                               *
*****************************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-AS400.
 OBJECT-COMPUTER.  IBM-AS400.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 old.
    05  oldname       PIC X(10).
    05  oldlibr       PIC X(10).
 77 scope             PIC X VALUE "P".
 01 errparm.
    05 input-l         PIC S9(6) BINARY VALUE ZERO.
    05 output-l        PIC S9(6) BINARY VALUE ZERO.
    05 exception-id    PIC X(7).
    05 reserved        PIC X(1).
    05 exception-data PIC X(50).
 01  new.
    05  newname       PIC X(10) VALUE "ACERRF24".
    05  newlibr       PIC X(10) VALUE "UTCBL".
 77  newlib           PIC X(10).
 PROCEDURE DIVISION.
 main-proc.
     DISPLAY "in ACF24".
     PERFORM variation-01 THRU end-variation.
     STOP RUN.
 variation-01.
*****************************************************************
*                                                               *
* This variation addresses the situation where there is no      *
* pending COBOL main, so no pending error handler can exist.    *
*                                                               *
*****************************************************************
     DISPLAY "no pending so expect nothing but error LBE7052".
     MOVE SPACES TO old exception-id.
*****************************************************************
* By setting error parm > 8, expect escape message              *
* LBE7052 to be returned in error parameter.                    *
*****************************************************************
     MOVE LENGTH OF errparm TO input-l.
     CALL "QLRRTVCE" USING old scope errparm.
     IF exception-id IS NOT =  "LBE7052" THEN
       DISPLAY "** error - expected LBE7052"
     ELSE
       DISPLAY "LBE7052 was found"
     END-IF.
*****************************************************************
* Reset input-l to ZERO,  thus any further errors will cause    *
* COBOL program to stop.                                        *
*****************************************************************
     MOVE 0 TO input-l.
     MOVE SPACES TO old exception-id.
 variation-02.
*****************************************************************
*                                                               *
* This variation creates a pending run unit.  It then makes     *
* sure that no pending error handler has been set.              *
*                                                               *
*****************************************************************
     DISPLAY "create pending run unit".
     CALL "QLRCHGCM" USING errparm.
*****************************************************************
*                                                               *
* No pending error handler exists so *NONE should be            *
* returned.                                                     *
*                                                               *
*****************************************************************
```

```
     CALL "QLRRTVCE" USING old scope errparm.
     DISPLAY "Retrieved Error Handler is=" old.
     IF oldname IS NOT = "*NONE" THEN
       DISPLAY "** error - expected *NONE for error handler"
     END-IF.
     MOVE 0 TO input-l.
     MOVE SPACES TO old exception-id.
 variation-03.
*************************************************************
*                                                           *
* This variation sets an error handler for the pending      *
* run unit and then does another check to make sure it      *
* was really set.                                           *
*                                                           *
*************************************************************
     CALL "QLRSETCE" USING new scope newlib old errparm.
     IF oldname IS NOT = "*NONE"
       DISPLAY "** error in oldname "
     END-IF.
     IF newlib IS NOT = "UTCBL"
       DISPLAY "** error in new library "
     END-IF.
*************************************************************
* Call the retrieve API to check to make sure that the      *
* set API worked.                                           *
*************************************************************
     MOVE SPACES TO old exception-id.
     CALL "QLRRTVCE" USING old scope errparm.
     DISPLAY "Retrieved Error Handler is=" old.
     IF oldname IS NOT = "ACERRF24" OR oldlibr IS NOT = "UTCBL"
       DISPLAY "** error - expected ACERRF24 error handler"
     END-IF.
 end-variation.
```

Error Handler for Example COBOL Program

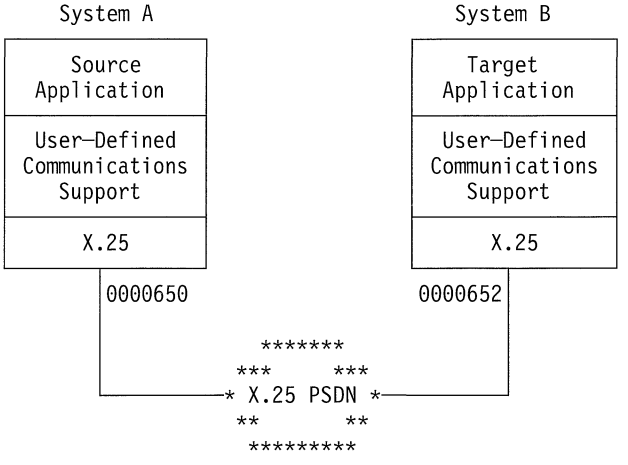
This example error handler works with "Using COBOL Program to Call APIs" on page A-33.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ACERRF24.
************************************************
* Error handler for program ACF24              *
************************************************
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-AS400.
 OBJECT-COMPUTER.  IBM-AS400.
 SPECIAL-NAMES. SYSTEM-CONSOLE IS SYSCON.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 scope PIC X VALUE "P".
 01 errparm.
    05 FILLER PIC X(30).
 LINKAGE SECTION.
 77  cobol-id PIC X(7).
 77  valid-responses PIC X(6).
 01  progr.
     05  progname PIC X(10).
     05  proglibr PIC X(10).
 77  system-id PIC X(7).
 77  len-text PIC S9(9) COMP-4.
 01  subtext.
     03  subchars PIC X OCCURS 1 TO 230 TIMES
         DEPENDING ON len-text.
 77  retcode PIC X(1).
 PROCEDURE DIVISION USING cobol-id, valid-responses,
          progr, system-id, subtext, len-text, retcode.
 main-proc.
*************************************************************
* check for typical messages and take appropriate action *
*************************************************************
     EVALUATE cobol-id
     WHEN "LBE7604"
*************************************************************
* stop literal, let the user see the message            *
*************************************************************
       MOVE SPACE TO retcode
     WHEN "LBE7208"
*************************************************************
* accept/display, recoverable problem answer G to continue
*************************************************************
       MOVE "G" TO retcode
     WHEN OTHER
*************************************************************
* for all other messages signal system operator and      *
*  end the current run unit                              *
*************************************************************
       DISPLAY "COBOL Error Handler ACERRF24 "
               "Found message " cobol-id
               " Issued from program " progr
               UPON syscon
       DISPLAY " Ended current run unit"
               UPON syscon
       MOVE "C" TO retcode
     END-EVALUATE.
     GOBACK.
```

Using the User-Defined Communications Programs for File Transfer

This section provides a simple example showing how X.25-oriented applications use the user-defined communications support to connect to remote systems. Two user-defined application programs written in the C/400 programming language are used to illustrate a simple file transfer between two systems over an X.25 packet-switching data network (PSDN). Although an X.25 example is shown, many of the same concepts can be applied to applications running over token-ring and Ethernet local area networks (LANs). For the purposes of the examples, the APIs are referred to by their call names. For complete information about the user-defined communications APIs, see Chapter 6, "User-Defined Communications Support APIs" on page 6-1.

For this example, the following network configuration will be used.



X.25 Overview

In this example X.25 network, the source application on System A is responsible for establishing a switched virtual circuit, or connection to the target application running on System B. This is done by using the remote network address (System B's address) of X'0000652'. When the target application on System B is initialized, it waits for notification of an incoming call packet before proceeding. Once

the virtual circuit is established, the source application reads records from a file into its output buffer and sends them to the target application using normal X.25 data transfer procedures. While receiving the file data, the target application writes the data to a local file on System B. When the file transfer completes, the source application closes the connection by issuing an X.25 clear request packet and ends. When receiving the clear indication packet, the target application also ends.

User-Defined Communications Support Overview

Both the source and target applications call the Query Line Description (QOLQLIND) API to obtain information about the local X.25 line being used. This information is stored in a local control block for use in establishing the peer connection during X.25 connection processing. Both applications also call the Enable Link (QOLELINK) API to enable the link for future communications. The AS/400 line name, communications handle, and remote DTE address are passed to both programs as arguments to the C function main(). For simplicity, the user space names and data queue name on the call to the QOLELINK API are coded directly in the applications.

Note: Keyed data queue support is used by both applications. The key length is 3 and the keys used are source and target for the source and target applications, respectively.

Activating Filters: Once the links have been enabled and both applications have read their respective enable-complete entries from their data queues, the target application program calls the Set Filter (QOLSETF) API to activate a filter. The filter activated then identifies the protocol of the local X.25 service user. This filter is used by the user-defined communications support on System B to route incoming calls. The actual filter type activated is X'00' (for X.25 PID) and its associated value is X'21'. For more information concerning filters, see "Set Filter (QOLSETF) API" on page 6-42. After activating the X'21' filter, the target application waits for the source application to request a connection.

Establishing a Connection: The source application calls the Send Data (QOLSEND) API with a X'B000' operation in its output data buffer to establish a switched virtual circuit (SVC) to the target application. Included in the first byte of the call user data is the protocol ID of the target application, or X'21'. When the user-defined communications support on System B sees the incoming call packet with the first byte of user data equal to a previously activated filter, the call is routed to the process responsible for activating that filter. In this case, the target application will receive notification of an incoming call since it previously activated filter X'21'.

While waiting for the incoming call, the target application calls the Receive Data (QOLRECV) API to receive a X'B201' operation with incoming call data. After doing so, the target application accepts the X.25 connection by calling the QOLSEND API with a X'B400' operation in its output data buffer. See "Send Data (QOLSEND) API" on page 6-26 for more information.

Sending Data: Once the peer connection is established between the source and target applications running on System A and System B respectively, the file transfer takes place. The source application reads records from a local file and calls the QOLSEND API with X'0000' operations in its output data buffer to transfer the file data to System B. This process continues until the entire contents of the source file has been sent to System B.

Receiving Data: After accepting the X.25 connection, the target application waits until its data queue receives incoming-data entries. When the first entry is read from the queue, the QOLRECV API is called to determine which operation was received. Barring failure, the target application should receive a X'0001' operation as a result of the QOLRECV API call. The data contained in the input data buffer is the file data received from System A. While receiving the file data, the target application writes the data to a local file. This process continues until the entire contents of the file is received from System A. The target application then assumes the file transfer is complete when an operation other than a X'0001' operation is received after a successful call to the QOLRECV API. Most likely, the first non-X'0001' operation received will be X'B301' operation, signalling that the user-defined communications support running on System B received an SVC clear indication.

Clearing the Connection and Disabling Links: Once the entire contents of the file has been read and sent to System B, the source application calls the QOLSEND API with a X'B100' operation in its output data buffer to clear the X.25 connection. Afterwards, the source application closes its local file, disables its local link by calling the QOLDLINK API, and ends.

When the source application program sends a X'B100' operation, it causes the target application to receive a X'B301' operation. After receiving this operation, the target application program calls the QOLSEND API with a X'B100' operation to locally close the connection between itself and the user-defined communications support. Afterwards, the target application closes its local file, disables its local link by calling the QOLDLINK API, and ends.

Using Timers and the Data Queue Support: Both the source and target application programs use the user-defined communications support timer service to manage the reception of certain operations. This is done by setting a timer before checking the data queue for an entry. For example, the target application sets a timer to manage the reception of file data from the source application. If the timer expires, the user-defined communications support places a timer-expired entry on the application's data queue. The target application then assumes when receiving this entry that the source application ended abnormally. The target application can then take the appropriate action to end itself.

C/400 Compiler Listings

Below are the listings for the source and target applications described in the previous paragraphs. Note the reference numbers (for example, 1) in the listings. Detailed explanations of each reference number block are found on pages A-44 and A-52.

The target application compiler listing can be found in "Target Application on System B Listing" on page A-45.

Source Application on System A Listing: In this example, the source application is the initiator of all meaningful work. In summary, the source program listed on the following pages does the following:

- Calls the QOLQLIND API to get local X.25 line information
- Opens the local file
- Calls the QOLELINK API to establish a link for communications
- Calls the QOLSEND API with X'B000' operation to establish a peer (SVC) connection
- Sends the local file to the target system via X'0000' operations
- Calls the QOLSEND API with X'B100' operation to clear the peer (SVC) connection
- Calls the QOLDLINK API to disable the link
- Calls the QOLTIMER API to manage the reception of data queue entries

```
Program name  . . . . . : SOURCE
  Library name  . . . . . :     UDCS_APPLS
Source file . . . . . :   QCSRC
  Library name  . . . . . :     UDCS_APPLS
Source member name  . . . . . :   SOURCE
Text Description  . . . . . :   Source Application Example
Compiler options  . . . . . :   *SOURCE  *NOXREF  *SHOWUSR
                              :   *SHOWSYS  *NOSHOWSKP  *NOEXPMAC
                              :   *NOAGR *NOPPONLY  *NODEBUG
                              :   *GEN *NOSECLVL  *PRINT  *LOGMSG
Language level options  . . . . :
Source margins:
  Left margin . . . . . :   1
  Right margin  . . . . . :   80
Sequence columns:
  Left Column . . . . . :
  Right Column  . . . . . :
Define name . . . . . :
Generation options  . . . . . :   *NOLIST *NOXREF *GEN  *NOATR
                              :   *NODUMP  *NOOPTIMIZE *NOALWBND
                              :   *NOANNO
Print file  . . . . . :   QSYSPRT
  Library name  . . . . . :     *LIBL
Message flagging level  . . . . :   0
Compiler message:
  Message limit . . . . . :   *NOMAX
  Message limit severity  . . . :   30
Replace program object  . . . . :   *YES
User profile  . . . . . :   *USER
Authority . . . . . :   *LIBCRTAUT
Target Release  . . . . . :   *CURRENT
INDEBUG options . . . . . :   I don't know
Last change . . . . . :   90/12/19 08:49:37
Source description  . . . . . :   Source Application Example
Compiler  . . . . . :   IBM SAA C/400 Compiler

/*****************************************************************/
/**  Program Name: Source Application Program Example           **/
/**                                                             **/
/**                                                             **/
/**  Function:                                                  **/
/**  This is the source application program example that uses   **/
/**  X.25 services provided by the user-defined communications  **/
/**  support to transfer a simple file to the target application **/
/**  program running on system B.  This program performs the    **/
/**  following:                                                 **/
/**     01.  Open the source file name INFILE.                  **/
/**     02.  Call QOLQLIND API to obtain local line information.**/
/**     03.  Enable a link.                                     **/
/**     04.  Send a 'B000'X operation (call request).           **/
/**     05.  Receive a 'B001'X operation (call confirmation).   **/
/**     06.  Read record(s) from the file opened in step 1). and**/
/**          send '0001'X operation(s) to transfer the file to  **/
/**          the target application program.                    **/
/**     07.  Send a 'B100'X operation (clear call request).     **/
/**     08.  Receive a 'B101'X operation.                       **/
/**     09.  Disable the link enabled in step 3).               **/
/**                                                             **/
/**  A data queue will be actively used to manage the operation **/
/**  of this program.  Data queue support will be used to monitor**/
/**  for the completion of the enable and disable routines, as  **/
/**  well as timer expirations and incoming data.  Timers are   **/
/**  used to ensure that there will never be an infinite wait on**/
/**  the data queue.  If a timer expires, the link enabled will **/
/**  be disabled and the program will stop.                     **/
/**                                                             **/
/** Inputs:                                                     **/
/** The program expects the following input parameters          **/
/**     Line Name:   This is the name of the line description   **/
/**                  that will be used to call the QOLELINK API.**/
/**                  The line must be an X.25 line with at least**/
/**                  one SVC of type *SVCBOTH or *SVCOUT.       **/
/**                                                             **/
/**     CommHandle:  This is the logical name that will be used **/
/**                  to identify the link enabled.              **/
/**                                                             **/
/**     Remote DTE Address:  The is the Local Network Address   **/
/**                          of System B.                       **/
/**                                                             **/
```

Figure A-1 (Part 1 of 17). C/400 Compiler Listing for the Source Application

```
/**                                                                **/
/** Outputs:                                                       **/
/** Current status of the file transfer will be provided when      **/
/** running this program.  If an error should occur, then a        **/
/** message will be displayed indicating where the error occurred  **/
/** and the program will end.        If the program completes      **/
/** successfully, a "successful completion" message will be        **/
/** posted.                                                        **/
/**                                                                **/
/********************************************************************/

#include "header"

#include "typedefs"

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <xxasio.h>
#include <xxcvt.h>
#include <string.h>
#include <ctype.h>
1

typedef struct queuein
    {
      char library??(10??);
      char name??(10??);
      char option;
    } queuein;

typedef struct namelib
    {
      char library??(10??);
      char name??(10??);
    } namelib;

typedef _Packed struct format1
  {
   char type;
   char reserved1;
   unsigned short logchanid;
   unsigned short sendpacksize;
   unsigned short sendwindsize;
   unsigned short recvpacksize;
   unsigned short recvwindsize;
   char reserved2??(7??);
   char dtelength;
   char dte??(16??);
   char reserved3??(8??);
   char dbit;
   char reserved4??(7??);
   char cug;
   char cugid;
   char reverse;
   char fast;
   char faclength;
   char facilities??(109??);
   char reserved5??(48??);
   unsigned short calllength;
   char callud??(128??);
   char reserved6??(128??);
   unsigned char misc??(4??);         /* control flags */
   unsigned int maxasmsize;
   unsigned short autoflow;
} format1;

typedef _Packed struct format2
  {
   unsigned short type;
   char cause;
   char diagnostic;
   char reserved??(4??);
   char faclength;
   char facilities??(109??);
   char reserved2??(48??);
   unsigned short length;
   char userdata??(128??);
  } format2;
```

Figure A-1 (Part 2 of 17). C/400 Compiler Listing for the Source Application

```
typedef _Packed struct desc
  {
   unsigned short length;
   char more;   /*These 4 char's are only used for X.25.*/
   char qualified;
   char interrupt;
   char dbit;
   char reserved??(26??);
  } desc;

typedef _Packed struct llcheader
  {
   unsigned short headerlength;
   char macaddr??(6??);
   char dsap;
   char ssap;
   char priority;
   char priorctl;
   unsigned short routlen;
   unsigned short userdtalen;
   char data??(1??);
  } llcheader;

typedef _Packed struct espec
  {
   char reserved??(2??);
   unsigned int hwecode;
   unsigned int timestamphi;
   unsigned int timestamplo;
   unsigned int elogid;
   char reserved2??(10??);
   char flags;
   char cause;
   char diagnostic;
   char reserved3;
   unsigned int erroroffset;
   char reserved4??(4??);
  } espec;

typedef struct tableentry
  {
    char handle??(10??);
    char type;
    char inbuff??(20??);
    char indesc??(20??);
    char outbuff??(20??);
    char outdesc??(20??);
    unsigned int totaldusize;
    struct tableentry *next;
  } tableentry;

/*******  Data structure for X.25 line    ********/
/**** descriptions as returned by QOLQLIND. ******/

typedef struct x25info
  {
   char addrlen;
   char addr??(9??);
   char addrtype;
   char insert;
   char modulus;
   char dtedce;
   unsigned short maxsend;
   unsigned short maxrecv;
   unsigned short defsend;
   unsigned short defrecv;
   char windowsend;
   char windowrecv;
   unsigned short numlc;
   char lcinfo??(4??);
  } x25info;
```

Figure A-1 (Part 3 of 17). C/400 Compiler Listing for the Source Application

```
typedef struct querydata
  {
   char header??(12??);    /* line header info */
   x25info x25data;        /* preliminary data */
  } querydata;
#include "hexconv"
#include <stdio.h>

char *inttohex(decimal,hex)  /*Converts a 4-byte integer
       into a string of 2 uppercase hex characters.*/
unsigned int decimal;
char *hex;

{
  sprintf(hex,"%.2X",decimal);
  return(hex);
}

unsigned int hextoint(hex)  /*Converts a string containing
                hex digits into a 4-byte integer.   */
char *hex;
{
  int decimal;

  sscanf(hex,"%x",&decimal);
  return(decimal);
}
```

2

```
#pragma linkage(QOLDLINK, OS)
#pragma linkage(QOLELINK, OS)
#pragma linkage(QOLSEND, OS)
#pragma linkage(QOLRECV, OS)
#pragma linkage(QUSPTRUS, OS)
#pragma linkage(QRCVDTAQ, OS)
#pragma linkage(QCLRDTAQ, OS)
#pragma linkage(QOLTIMER, OS)
#pragma linkage(QOLSETF, OS)
#pragma linkage(QOLQLIND, OS)

FILE *screen;
FILE *rptr;
FILE *fptr;

extern void QOLDLINK(int *, int *, char *, char *);
```

3

```
extern void QOLELINK (int *, int *, int *, int *, int *, int *,\
                      char *, char *, char *, char *, int *, char *,\
                       char *, char *, char *);

extern void QOLSEND (int *, int *, void *, int *, int *, int *,\
                     char *, unsigned short *, int *);

extern void QOLRECV (int *, int *, int *, int *, unsigned short *,\
                     int *, char *, void *, char *);

extern void QOLSETF (int *, int *, int *, char *);

extern void QOLTIMER (int *, int *, char *, char *, char *, char *,
                      int *, int *, int *, char *, char *);

extern void QUSPTRUS (void *, void *);

extern void QRCVDTAQ (char *, char *, char *, void *, char *,\
                      char *, char *, char *, char *, char *);

extern void QCLRDTAQ (char *, char *);

extern void QOLQLIND(int *, int *, int *, void *, char *, char *);
```

Figure A-1 (Part 4 of 17). C/400 Compiler Listing for the Source Application

```
/************   Typedef  Declarations      ******************/

typedef struct usrspace
    {
      char name??(10??);
      char library??(10??);
    } usrspace;
```

4

```
typedef struct enableparms     /*  Enable parameters  */
  {
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       tdusize,             /*  Output  */
       numunits,            /*  Output  */
       maxdtalan,           /*  Output  */
       maxdtax25,           /*  Input   */
       keylength;           /*  Input   */
char keyvalue??(256??),     /*  Input  */
     linename??(10??);      /*  Input  */
  } enableparms;

typedef struct disableparms     /*  Disable parameters  */
  {
  int retcode,             /*  Output  */
      reason;              /*  Output  */
  char vary;               /*  Input  */
  } disableparms;

typedef struct setfparms   /*  Set Filters parameters  */
  {
  int retcode,             /*  Output  */
      reason,              /*  Output  */
      erroffset;           /*  Output  */
  } setfparms;

typedef _Packed struct hdrparms /* Filter header */
  {
   char function;
   char type;
   unsigned short number;
   unsigned short length;
   char filters??(1??);
  } hdrparms;

typedef _Packed struct x25filter /* X.25 filter */
  {
   char pidlength;
   char pid;
   char dtelength;
   char dte??(12??);
   char flags;
  } x25filter;

typedef struct sendparms     /*  Send parameters  */
  {
 espec errorspecific;       /*  Output  */
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       newpcep,             /*  Output  */
       ucep,                /*  Input  */
       pcep,                /*  Input   */
       numdtaelmnts;        /*  Input   */
unsigned short operation;   /*  Input  */
  } sendparms;
```

Figure A-1 (Part 5 of 17). C/400 Compiler Listing for the Source Application

```
typedef struct recvparms   /*  Receive parameters  */
  {
 espec errorspecific;       /*  Output  */
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       newpcep,             /*  Output  */
       ucep,                /*  Output  */
       pcep,                /*  Output   */
       numdtaunits;         /*  Output   */
  char dataavail;           /*  Output  */
unsigned short operation;   /*  Output  */
  } recvparms;

typedef struct timerparms   /*  Timer parameters  */
  {
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       interval,            /*  Input  */
       establishcount,      /*  Input  */
       keylength;           /*  Input   */
  char handleout??(8??),    /*  Output  */
       handlein??(8??),     /*  Input  */
       operation,           /*  Input  */
       keyvalue??(256??),   /*  Input  */
       userdata??(60??);    /*  Input  */
  } timerparms;

typedef struct especout
  {
   char hwecode??(8??);
   char timestamp??(16??);
   char elogid??(8??);
   char fail;
   char zerocodes;
   char qsysopr;
   char cause??(2??);
   char diagnostic??(2??);
   char erroffset??(6??);
  } especout;

typedef struct qlindparms     /*  Query line parameter
  {
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       nbytes;              /*  Output  */
   char userbuffer??(256??);
   char format;
  } qlindparms;

typedef _Packed union content       /*  Queue support
  {
    _Packed struct other
      {
       char commhandle??(10??);
       char reserved??(58??);
      } other;
    _Packed struct enable
      {
       char commhandle??(10??);
       char status;
       char reserved??(57??);
      } enable;
    _Packed struct timer
      {
       char timerhandle??(8??);
       char userdata??(60??);
      } timer;
  } content;
```

Figure A-1 (Part 6 of 17). C/400 Compiler Listing for the Source Application

```
typedef _Packed struct qentry  /*  Queue  parameter */
  {
    char type??(10??);
    unsigned short msgid;
    content message;
    char key??(256??);
  } qentry;
```

5

```
void senddata(sendparms *a, char *b, desc *c, char *d, char *e, int f);
void sndformat1(sendparms *a,char *b, char *c, char *d, qlindparms *f);
void sndformat2 (sendparms *a, char *b, char *c);
void setfilters (hdrparms *a);
void byte (char *a, int b, char *c, int d);
void printespec (espec *a);
void settimer(unsigned short *a,char *b,qentry *c,usrspace *d,char *e);
void dequeue (int a, char *b, qentry *c, usrspace *d);
void x25lind (qlindparms *a, char *b);
int getline (char *a, int b, FILE *c);
void disablelink (disableparms *a, char *b, usrspace *c);
void handler (disableparms a, usrspace *b);
sigdata_t *sigdata(void);

/*********************************************************/
/**************   Start Main Program   ******************/
/*********************************************************/

main (int argc, char *argv??(??))
{

/************  Variable  Declarations      ******************/

  usrspace inbuff,       /*  Input Data Buffer */
           indesc,       /*  Input Buffer Descriptor  */
           outbuff,      /*  Output Data Buffer  */
           outdesc,      /*  Output Buffer Descriptor */
           qname;        /*  Data Queue  */

  int length,                 /* Data Queue key length  */
      linesiz,                /* Length of line that is read in */
      i= 0;                   /* counter  */
  unsigned short expctid;     /*  Message ID that is expected */
  char commhandle??(10??),    /*  Command Line Parameter  */
       *buffer,               /* Pointer to buffer     */
       rmtdte??(18??),        /* Remote DTE read in       */
       line??(132??),         /* Line to read in       */
       key??(256??);          /* Data Queue key identifier  */
  desc *descriptor;           /* Pointer to buffer descriptor  */

  /** definitions for the API functions  **/
  enableparms enable;
  disableparms disable;
  sendparms send;
  recvparms recv;
  setfparms setf;
  timerparms timer;
  qlindparms qlind;
  qentry dataq;
  hdrparms *header;
```

6

```
  /***---  Open the file to send to remote side       ----**/
  if ((fptr = fopen("UDCS_APPLS/INFILE(INFILE))", "r")) == N
     {
     printf("Unable to open source input file in UDCS_APPLS LIB.\n"
     printf("The Program was terminated.\n\n");
     return;
     }
  /***---  Open the display file as our input screen.  ----**/
  if ((screen = fopen("ERRORSPEC", "ab+ type=record")) == NULL)
     {
     printf("Unable to open display file.\n");
     printf("The Program was terminated.\n\n");
     return;
     }
```

Figure A-1 (Part 7 of 17). C/400 Compiler Listing for the Source Application

```
/** set the exception handler  **/
signal(SIGABRT,&handler);

/**  Clear the command line Parameters  **/
strncpy(enable.linename, "          ", 10);  /* Clear linename */
strncpy(commhandle, "          ", 10);       /* Clear Commhandle*/
strncpy(rmtdte, "                ", 17);     /* Clear Remote DTE*/

/**  Receive command line Parameters  **/
strncpy(enable.linename, argv??(1??), strlen(argv??(1??)));
strncpy(commhandle, argv??(2??), strlen(argv??(2??)));
strncpy(rmtdte, argv??(3??), strlen(argv??(3??)));
rmtdte??(strlen(argv??(3??))??) = '\0';

/** Initialize the user spaces  **/
strncpy(inbuff.library, "UDCS_APPLS", 10);    /*  Input Buffer */
strncpy(inbuff.name, "SOURCEIBUF", 10);
strncpy(indesc.library, "UDCS_APPLS", 10);    /* Input B Desc  */
strncpy(indesc.name, "SOURCEBDSC", 10);
strncpy(outbuff.library, "UDCS_APPLS", 10);   /*  Output Buffer*/
strncpy(outbuff.name, "SOURCEOBUF", 10);
strncpy(outdesc.library, "UDCS_APPLS", 10);   /* Output B Desc */
strncpy(outdesc.name, "SOURCEODSC", 10);
strncpy(qname.library, "UDCS_APPLS", 10);   /*  Data queue  */
strncpy(qname.name, "X25DTAQ   ", 10);

/*****   retrieve the line description information  ******/
x25lind (&qlind, enable.linename);

if ((qlind.retcode != 0) || (qlind.reason != 0))
   {
   printf("Query line description failed.\n");
   printf("Return code = %d\n", qlind.retcode);
   printf("Reason code = %d\n\n", qlind.reason);
   return;
   }

/*****   Hard Code the QOLELINK Input Parameters  ******/
enable.maxdtax25 = 512;
enable.keylength = 3;
strncpy (enable.keyvalue, "SND", 3);
```

7

```
/************************************************/
/************  Enable the line  *****************/
/************************************************/
QOLELINK (&(enable.retcode), &(enable.reason), &(enable.tdusize),\
    &(enable.numunits), &(enable.maxdtalan), &(enable.maxdtax25),\
    (char *)&inbuff, (char *)&indesc, (char *)&outbuff,\
    (char *)&outdesc, &(enable.keylength), enable.keyvalue,\
    (char *)&qname, enable.linename, commhandle);

if ((enable.retcode != 0) || (enable.reason != 0))
   {
   printf("Line %.10s with Commhandle %.10s was NOT ENABLED.\n",\
           enable.linename, commhandle);
   printf("Return code = %d\n", enable.retcode);
   printf("Reason code = %d\n\n", enable.reason);
   return;
   }
```

8

```
/*-------- Set a timer for Enable Link  ---------**/
expctid = 0xF0F0;
settimer(&expctid, "Enable", &dataq, &qname, commhandle);
if (expctid != 0xF0F0)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }
```

Figure A-1 (Part 8 of 17). C/400 Compiler Listing for the Source Application

9

```
/*****************************************************************/
/*************   Set up a Call Request Packet  ******************/
/*****************************************************************/


/****   Get pointers to the user spaces.   ******/
QUSPTRUS(&outbuff, &buffer);
QUSPTRUS(&outdesc, &descriptor);

send.ucep = 26;              /*  set the UCEP number  */
send.operation = 0xB000;     /*  send a call request  */
send.numdtaelmnts = 1;       /*  send one data unit   */

/**-----------    Send the packet   ---------**/
sndformat1 (&send, buffer, rmtdte, commhandle, &qlind);

if ((send.retcode != 0) || (send.reason != 0))
   {
   printf("Call request packet not sent\n");
   printf("Return code = %d\n", send.retcode);
   printf("Reason code = %d\n", send.reason);
   printf("new pcep %d\n", send.newpcep);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }
```

10

```
/*****************************************************************/
/***********   Receive the Call CONFIRMATION packet       ********/
/*****************************************************************/

/*-------- Set a timer to receive a message  ---------**/
expctid = 0xF0F3;
settimer(&expctid, "Rcv Call", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }

/** Get pointer to use space  **/
QUSPTRUS (&inbuff, &buffer);
QUSPTRUS (&indesc, &descriptor);

QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
         &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
         &(recv.dataavail), &(recv.errorspecific), commhandle);

if ((recv.retcode != 0) || (recv.reason != 0))
   {
   printf("Recv Call reqst resp failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

/* Interpret the Received Operation  */
if (recv.operation != 0xB001)
   {
   printf("Recvd opr %x instead of opr B001\n", recv.operation);
   disablelink (&disable, commhandle, &qname);
   return;
   }

printf("We have an X.25 SVC connection\n\n");
```

Figure A-1 (Part 9 of 17). C/400 Compiler Listing for the Source Application

11

```
/*****************************************************************/
/***************   Send the file to the target application *******/
/*****************************************************************/

send.pcep = send.newpcep;       /*  set the PCEP number  */

/***************   Send the Mbr LGRF in file DOC  ****************/
linesiz = getline(line, 92, fptr);   /* Get first record  **/
while (linesiz != 0)
   {
   /***************   Send a Packet of Data    ***************/

   /****   Get pointers to the user spaces.   ******/
   QUSPTRUS(&outbuff, &buffer);
   QUSPTRUS(&outdesc, &descriptor);

   send.operation = 0x0000;
   send.numdtaelmnts = 1;

   /**-----   Send the packet     -------------**/
   senddata (&send, buffer, descriptor, commhandle, line, linesiz);

   if ((send.retcode != 0) || (send.reason != 0))
      {
      printf("Data NOT sent for commhandle %.9s\n", commhandle);
      printf("Return code = %d\n", send.retcode);
      printf("Reason code = %d\n", send.reason);
      printf("new pcep %d\n", send.newpcep);
      printespec(&(send.errorspecific));

      disablelink (&disable, commhandle, &qname);
      return;
      }
   i = i + 1;
   printf("Data %d Sent for commhandle %.9s.\n\n", i, commhandle);

   linesiz = getline(line, 92, fptr);  /** Get next record  **/
   }                          /***  End While loop  ***/


/*****************************************************************/
/***************   Set up a Clear Request Packet  ***************/
/*****************************************************************/
```

12

```
/****   Get pointers to the user spaces.   ******/
QUSPTRUS(&outbuff, &buffer);
QUSPTRUS(&outdesc, &descriptor);

send.operation = 0xB100;          /**  send clear request  **/
send.numdtaelmnts = 1;            /**  send one data unit  **/

/**-----------    Send the packet   ---------**/
sndformat2 (&send, buffer, commhandle);

if ((send.retcode != 0) || (send.reason != 0))
   {
   printf("Clear request packet not sent\n");
   printf("Return code = %d\n", send.retcode);
   printf("Reason code = %d\n", send.reason);
   printf("new pcep %d\n", send.newpcep);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }
```

Figure A-1 (Part 10 of 17). C/400 Compiler Listing for the Source Application

13

```
/*****************************************************************/
/***********   Receive the Clear Request Response packet   *****/
/*****************************************************************/

/*-------- Set a timer to receive a message  ---------**/
expctid = 0xF0F3;
settimer(&expctid, "Rv Clr Rqt", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }

/***********   Call QOLRECV to Receive the Clear Response  *****/
/****   Get pointers to the user spaces.   ******/
QUSPTRUS (&inbuff, &buffer);
QUSPTRUS (&indesc, &descriptor);

QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
         &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
         &(recv.dataavail), &(recv.errorspecific), commhandle);

if ((recv.retcode != 0) || (recv.reason != 0))
   {
   printf("Recv clear response failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

/* Interpret the Received Operation  */
if (recv.operation != 0xB101)
   {
   printf("Recvd opr %x instead of opr B101\n", recv.operation);
   disablelink (&disable, commhandle, &qname);
   return;
   }

/*************************************************/
/***  Disable the link and end the program  ****/
/*************************************************/
disablelink (&disable, commhandle, &qname);

printf("******   SSNDTH completed successfully   ******\n\n");

}    /* End Main */

/*****************************************************************/
/**************** Start Subroutine Section    ********************/
/*****************************************************************/

/*********************************************************/
/***************     Send a Packet of Data  *****************/
```

14

```
void senddata (sendparms *send,
               char *buffer,
               desc *descriptor,
               char *commhandle,
               char *line,
               int linesiz)

{
  descriptor->length = linesiz;
  descriptor->more = 0;
  descriptor->qualified = 0;
  descriptor->interrupt = 0;
  descriptor->dbit = 0;
  strncpy (buffer, line, linesiz);

  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep), \
           commhandle, &(send->operation), &(send->numdtaelmnts));

}  /* End senddata Subroutine  */
```

Figure A-1 (Part 11 of 17). C/400 Compiler Listing for the Source Application

```
/*********************************************************/
/************  Routine to fill X.25 Format I   ***********/

void sndformat1 (sendparms *send,
                 char *buffer,
                 char *rmtdte,
                 char *commhandle,
                 qlindparms *qlind)

{
  format1 *output = (format1 *) buffer;
  register int counter;
  register querydata *qd;

  qd = (querydata *)&(qlind->userbuffer);
  output->type = 2;                 /*  SVC used  */
  output->logchanid = 0x0;
  output->sendpacksize = qd->x25data.defsend;
  output->sendwindsize = qd->x25data.windowsend;
  output->recvpacksize = qd->x25data.defrecv;
  output->recvwindsize = qd->x25data.windowrecv;

  output->dtelength = strlen(rmtdte);
  byte(output->dte, 16, rmtdte, strlen(rmtdte));
  output->dbit = 0;
  output->cug = 0;
  output->cugid = 0;
  output->reverse = 0;
  output->fast = 0;
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->calllength = 1;
  byte(output->callud, 128, "21", 2);   /* Contains Remote PID */
  output->misc??(0??) = 0;     /* change to 0x80 for reset support */
  output->misc??(1??) = 0;
  output->misc??(2??) = 0;
  output->misc??(3??) = 0;
  output->maxasmsize = 16383;
  output->autoflow = 32;


  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));

}  /* End sndformat1 Subroutine  */


/*********************************************************/
/************  Routine to fill X.25 Format II   ***********/

void sndformat2 (sendparms *send,
                 char *buffer,
                 char *commhandle)

{
  format2 *output = (format2 *) buffer;

  output->type = 1;
  output->cause = 'FF';
  output->diagnostic = 'FF';
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->length = 0;
  byte(output->userdata, 128, "", 0);


  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));

}  /* End sndformat2 Subroutine  */
```

Figure A-1 (Part 12 of 17). C/400 Compiler Listing for the Source Application

15

```
/*****************************************/
/********    Routine to disable   ***********/

void disablelink (disableparms *disable,
                  char *commhandle,
                  usrspace *qname)

{
unsigned short expctid;
qentry dataq;

  disable->vary = 1;     /*  Hard coded to be varied off  */

  QOLDLINK (&(disable->retcode), &(disable->reason),\
                       commhandle, &(disable->vary));

  if ((disable->retcode != 0) && (disable->reason != 00))
     {
     printf ("Link %.10s did not disabled.\n", commhandle);
     printf ("return code = %d\n", disable->retcode);
     printf ("reason code = %d\n\n", disable->reason);
     }

  /**-------  Set a timer to receive disable complete msg --------**/
  expctid = 0xF0F1;
  settimer(&expctid, "Disable", &dataq,  qname, commhandle);
  if (expctid != 0xF0F1)
     {
     printf("Disable link did not complete successfully");
     return;
     }

  printf ("%.10s link disabled \n", commhandle);

  /**  close the files  **/
  fclose(fptr);
  fclose(screen);

}  /* End disablelink Subroutine  */


/*********************************************************/
/**    Routine to convert string to Hexadecimal format   ******/

void byte (char *dest,
           int dlength,
           char *source,
           int slength)

{
  register int counter;
  char holder??(2??);

  for (counter=0;counter<dlength;counter++)
   dest??(counter??)=0;
  for (counter=slength-1;counter>=0;counter--)
   if (isxdigit(source??(counter??)))
     {
      holder??(0??)=source??(counter??);
      holder??(1??)='\0';
      if  (counter % 2 == 0)
        dest??(counter/2??) += (char) hextoint(holder)*16;
      else dest??(counter/2??) += (char) hextoint(holder);
     }

}  /* End byte Subroutine  */
```

Figure A-1 (Part 13 of 17). C/400 Compiler Listing for the Source Application

```
/*************************************************************/
/**    Routine to display the ErrorSpecific output       ******/

void printespec(espec *errorspecific)

{
  especout outparms;

  QXXFORMAT(screen, "ERRORSPEC ");
  sprintf(outparms.hwecode, "%.8X", errorspecific->hwecode);
  sprintf(outparms.timestamp, "%.8X%.8X", errorspecific->timestamphi,\
                                    errorspecific->timestamplo);
  sprintf(outparms.elogid, "%.8X", errorspecific->elogid);
  if (errorspecific->flags & 0x40)
    outparms.fail = 'Y';
  else outparms.fail = 'N';
  if (errorspecific->flags & 0x20)
    outparms.zerocodes = 'Y';
  else outparms.zerocodes = 'N';
  if (errorspecific->flags & 0x10)
    outparms.qsysopr = 'Y';
  else outparms.qsysopr = 'N';
  sprintf(outparms.cause,"%.2X", errorspecific->cause);
  sprintf(outparms.diagnostic, "%.2X", errorspecific->diagnostic);
  sprintf(outparms.erroffset, "%.6d", errorspecific->erroroffset);
  fwrite(&outparms, 1, sizeof(especout), screen);
  fread("", 0, 0, screen);

}  /* End printespec Subroutine  */

/*************   Set a timer and dequeue next entry  ******/
16

void settimer (unsigned short *expctid,
               char *process,
               qentry *dataq,
               usrspace *qname,
               char *commhandle)

{
timerparms timer;
disableparms disable;
int length;
char key??(6??);

  timer.interval = 20000;      /*  Set timer for 20 seconds */
  timer.establishcount = 1;
  timer.keylength = 3;          /*  Set key value  */
  strncpy(timer.keyvalue, "SRC", 3);
  timer.operation = 1;          /*  Set a timer  */

  QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
          timer.handlein, (char *)qname, &(timer.operation),\
          &(timer.interval), &(timer.establishcount),\
          &(timer.keylength), timer.keyvalue, timer.userdata);

  if ((timer.retcode != 0) || (timer.reason != 0))
     {
     printf("%s timer failed while being set.\n", process);
     printf("Return code = %d\n", timer.retcode);
     printf("Reason code = %d\n\n", timer.reason);
     }

/**-------  Dequeue an entry  --------**/
  strncpy(key, "SRC",3);
  length = 3;
  dequeue (length, key, dataq, qname);
```

Figure A-1 (Part 14 of 17). C/400 Compiler Listing for the Source Application

```
  /*** Cancel timer  ***/
  if (dataq->msgid != 0xF0F4)
     {
     strncpy(timer.handlein, timer.handleout, 8);
     timer.operation = 2;          /*  Set one timer  */

     QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
          timer.handlein, (char *)qname, &(timer.operation),\
          &(timer.interval), &(timer.establishcount),\
          &(timer.keylength), timer.keyvalue, timer.userdata);

     if ((timer.retcode != 0) || (timer.reason != 0))
        {
        printf("%s timer failed while being canceled\n", process);
        printf("Return code = %d\n", timer.retcode);
        printf("Reason code = %d\n\n", timer.reason);
        }
     }

  if (dataq->msgid != *expctid)
      {
      printf ("A %.4X message ID was received instead of %.4X\n",\
              dataq->msgid, *expctid);
      printf ("%s completion message was not received\n", process);
      *expctid = dataq->msgid;
      }

}  /* End settimer Subroutine  */

/*************************************************************/
/*******   Dequeues the Incoming Message and processes it ******/

void dequeue (int length,
              char *key,
              qentry *dataq,
              usrspace *qname)

{
  char fldlen??(3??),
       waittime??(3??),
       keylen??(2??),
       senderid??(2??),
       *pointer,
       order??(2??);
  register int counter;


  waittime??(0??) = 0;
  waittime??(1??) = 0;
  waittime??(2??) = 0x1D;   /*  Hard code a delay of infinite   */
  keylen??(0??) = 0;
  keylen??(1??) = 0x3F;   /* Hard code a keylength of 3  */
  senderid??(0??) = 0;
  senderid??(1??) = 0x0F;
  strncpy(order, "EQ", 2);

  fflush(stdin);
  pointer = (char *)dataq;
  for (counter = 0; counter < 336; counter++)
    pointer??(counter??) = 0;

  strncpy (dataq->type, "       ", 7);
  while ((strncmp(dataq->type, "*USRDFN", 7) != 0) || (fldlen == 0))
     QRCVDTAQ(qname->name, qname->library, fldlen, dataq, waittime,\
          order, keylen, key, senderid,"");

}  /* End dequeue Subroutine  */
```

Figure A-1 (Part 15 of 17). C/400 Compiler Listing for the Source Application

```
/************************************************************/
/**  x25lind:  Read a record into buf and return length   **/
17

void x25lind (qlindparms *qlind, char *linename)
{
register int counter;

  for(counter=0;counter<256;counter++)
     qlind->userbuffer??(counter??)=0;

  qlind->format = 0x01;
  QOLQLinD (&(qlind->retcode), &(qlind->reason), &(qlind->nbytes),\
            qlind->userbuffer, linename, &(qlind->format));

}  /*  End x25lind Subroutine  */

/*****************************************************/
/**  Getline:  Read a record into line and return length   **/

int getline (char *line, int max, FILE *fptr)
{
   if (fgets(line, max, fptr) == NULL)
      return 0;
   else
      return strlen(line);

}  /* End getline Subroutine  */

/*****************************************************/
/*  Exception handler, so that a failure will not
    kill the program, and any associated data!!  */
18

void handler (disableparms disable, usrspace *qname)
{
  sigdata_t *data;

  disablelink(&disable, "*ALL      ", qname);
  printf("The program received an excecption.\n");
  printf("Disable Link was called & the program was terminated.\n\n");

  data=sigdata();
  data->sigact->xhalt=0;
  data->sigact->xrtntosgnler=0;
  data->sigact->xresigprior=0;
  data->sigact->xresigouter=0;

}  /*  End handler Subroutine  */

         * * * * *  E N D  O F  S O U R C E  * * * * *


             * * * * *  I N C L U D E S  * * * * *
Include Name        Last change        Actual Include Name
header              90/12/19 08:49:31  UDCS_APPLS/QCSRC/HEADER
typedefs            90/12/19 08:49:31  UDCS_APPLS/QCSRC/TYPEDEFS
stdio.h             90/11/12 17:24:55  QCC/H/STDIO
stddef.h            90/11/12 17:24:54  QCC/H/STDDEF
errno.h             90/11/12 17:24:49  QCC/H/ERRNO
signal.h            90/11/12 17:24:53  QCC/H/SIGNAL
ctype.h             90/11/12 17:24:49  QCC/H/CTYPE
stdarg.h            90/11/12 17:24:54  QCC/H/STDARG
stdlib.h            90/11/12 17:24:56  QCC/H/STDLIB
signal.h            90/11/12 17:24:53  QCC/H/SIGNAL
xxasio.h            90/11/12 17:24:57  QCC/H/XXASIO
xxcvt.h             90/11/12 17:24:58  QCC/H/XXCVT
string.h            90/11/12 17:24:56  QCC/H/STRING
ctype.h             90/11/12 17:24:49  QCC/H/CTYPE
hexconv             90/12/19 08:49:27  UDCS_APPLS/QCSRC/HEXCONV
stdio.h             90/11/12 17:24:55  QCC/H/STDIO
           * * * * *  E N D  O F  I N C L U D E S  * * * * *
```

Figure A-1 (Part 16 of 17). C/400 Compiler Listing for the Source Application

```
                 * *  M E S S A G E  S U M M A R Y  * *
Total   Info    Warning    Error    Severe    Terminal
       (0-4)    (5-19)   (20-29)   (30-39)    (40-99)
  0      0        0         0         0          0
      * *  E N D  O F  M E S S A G E  S U M M A R Y  * *


   IBM SAA C/400   UDCS_APPLS/SOURCE
ROUTINE          BLOCK NUMBER  SCOPE  TYPE
<MAIN>                  2      LOCAL  MAIN-PROGRAM
__sigdata               6      LOCAL  PROCEDURE
__sgnl                 12      LOCAL  PROCEDURE
__frdinit              49      LOCAL  PROCEDURE
__getrec               50      LOCAL  PROCEDURE
__putrec               51      LOCAL  PROCEDURE
__frdexit              52      LOCAL  PROCEDURE
__fwrinit              53      LOCAL  PROCEDURE
__fwrexit              54      LOCAL  PROCEDURE
QXXFORMAT             129      LOCAL  PROCEDURE
__strlen              164      LOCAL  PROCEDURE
__strncmp             168      LOCAL  PROCEDURE
__strncpy             170      LOCAL  PROCEDURE
inttohex              181      ENTRY  PROCEDURE
hextoint              182      ENTRY  PROCEDURE
senddata              196      ENTRY  PROCEDURE
sndformat1            197      ENTRY  PROCEDURE
sndformat2            198      ENTRY  PROCEDURE
byte                  200      ENTRY  PROCEDURE
printespec            201      ENTRY  PROCEDURE
settimer              202      ENTRY  PROCEDURE
dequeue               203      ENTRY  PROCEDURE
x25lind               204      ENTRY  PROCEDURE
getline               205      ENTRY  PROCEDURE
disablelink           206      ENTRY  PROCEDURE
handler               207      ENTRY  PROCEDURE
main                  208      ENTRY  PROCEDURE
* * * * *  E N D  O F  C O M P I L A T I O N  * *
```

Figure A-1 (Part 17 of 17). C/400 Compiler Listing for the Source Application

The following reference numbers and explanations correspond to the reference numbers in the source application's program listing.

1 Some general C structure declarations used by both the source and target application programs.

Note: This example program was developed using a 5250 display station and keyboard where the left ([) and right (]) bracket characters are not supported. As a result of this, C language array declarations used in the applications use the character sequence of "??(" to denote a left bracket and "??)" to denote a right bracket.

You do not need to change C language applications containing array declarations with the bracket characters in order to compile and run on the AS/400.

2 C/400 compiler directives are used here to indicate that standard OS/400 linkage conventions should be used when calling the user-defined communications support APIs.

3 C external function definitions for the user-defined communications support APIs. Note that all parameters are passed by reference.

4 More C structure declarations that are used when calling the user-defined communications support APIs.

5 Function prototypes of the internal functions used in this program.

6 Call the C library routines fopen() and signal() to open the source file and set up a signal handler to process AS/400 exceptions, respectively. An example of an

exception would be accessing a data area with a NULL pointer. If an exception situation is encountered, the handler() will be called in order for the program to end.

7 Call the QOLQLIND API to retrieve local configuration information from the AS/400 line description about that will be used for communications. Next, call the QOLELINK API to enable the line description using the line name and communications handle passed as input parameters to this program.

8 Call the QOLTIMER API to time the completion of the enable link operation. If the timer expires before the enable-complete entry is posted on the this program's data queue, then this program will end.

9 Call the QOLSEND API with a X'B000' operation to establish a connection to the target application program.

10 Monitor the source program's data queue for the call confirmation. The source program will be notified of the call confirmation by call the QOLRECV API and receiving a X'B001' operation in the program's input buffer.

11 This is the main send loop for the source program. The data from the source file is placed one line at a time in the output buffer and then the QOLSEND API is called to send one data unit of the file to System B. This process repeats until the contents of the entire file have been transmitted to the target application.

12 Call the QOLSEND API with a X'B100' operation to clear the peer connection.

13 The source program will check its data queue for a response to the clear packet sent to the target system. Once the response is received, the program will clean up, call the QOLDLINK API to disable the link previously enabled, and end.

14 The following C functions illustrate the various user-defined communications support APIs.

15 This procedure illustrates a call to the QOLDLINK API. Note the vary option is set to vary off the associated AS/400 *USRDFN network device.

16 The settimer() calls the QOLTIMER API requesting timers for 20000 milliseconds, or twenty seconds. After setting a timer, the settimer() will call the dequeue() to remove an entry from the program's data queue.

17 The x25lind() illustrates calling the QOLQLIND API.

18 As mentioned in block 6, the handler() will be called when OS/400 exception situation is encountered. This function performs final processing, calls the QOLDLINK API, and ends the source application program.

Target Application on System B Listing: The target application waits for the source application to initiate the file transfer. The following list summarizes the actions of the target application:

- Calls the QOLQLIND API to get local X.25 line information
- Opens the local file
- Calls the QOLELINK API to establish a link for communications
- Calls the QOLSETF API to activate an X.25 protocol ID filter
- Calls the QOLRECV API to receive the X'B201' operation (incoming call)
- Calls the QOLSEND API with a X'B400' operation to accept the SVC connection
- Receives the file from the target system via X'0001' operations
- Calls the QOLRECV API to receive the X'B301' (connection failure notification)
- Call the QOLSEND API with 'B100' operation to locally close the SVC connection
- Calls the QOLDLINK API to disable the link
- Calls the QOLTIMER API to manage the reception of data queue entries

Explanations of the reference numbers in the listing can be found on page A-52.

```
Program name  . . . . . : TARGET
  Library name  . . . . . :     UDCS_APPLS
Source file . . . . . :   QCSRC
  Library name  . . . . . :     UDCS_APPLS
Source member name  . . . . . :   TARGET
Text Description  . . . . . :  Target Application Example
Compiler options  . . . . . :   *SOURCE  *NOXREF *NOSHOWUSR
                          :   *NOSHOWSYS *NOSHOWSKP *NOEXPMAC
                          :   *NOAGR *NOPPONLY *NODEBUG
                          :   *GEN  *NOSECLVL  *PRINT  *LOGMSG
Language level options  . . . . :
Source margins:
  Left margin . . . . . :   1
  Right margin  . . . . . :   80
Sequence columns:
  Left Column . . . . . :
  Right Column  . . . . . :
Define name . . . . . :
Generation options  . . . . . :   *NOLIST  *NOXREF  *GEN *NOATR
                          :   *NODUMP  *NOOPTIMIZE  *NOALWBND
                          :   *NOANNO
Print file  . . . . . :   QSYSPRT
  Library name  . . . . . :     *LIBL
Message flagging level  . . . . :   0
Compiler message:
  Message limit . . . . . :   *NOMAX
  Message limit severity  . . . :   30
Replace program object  . . . . :   *YES
User profile  . . . . . :   *USER
Authority . . . . . :   *LIBCRTAUT
Target Release  . . . . . :   *CURRENT
INDEBUG options . . . . . :   I don't know
Last change . . . . . :   90/12/19 08:49:37
Source description  . . . . . :   Target Application Example
Compiler  . . . . . :   IBM SAA C/400 Compiler
```

```
/*****************************************************************/
/**                                                             **/
/**  Program Name: Target Application Program Example           **/
/**                                                             **/
/**                                                             **/
/**  Function:                                                  **/
/**  This is the target application program example that uses   **/
/**  X.25 services provided by the user-defined communications  **/
/**  support to receive a simple file from the source application **/
/**  program running on System A.  This program performs the    **/
/**  following:                                                 **/
/**     01.  Open the target file named OUTFILE.                **/
/**     02.  Call QOLQLIND to obtain local line information.    **/
/**     03.  Enable a link.                                     **/
/**     04.  Set a Filter on the enabled link.                  **/
/**     05.  Receive a 'B101'X operation (incoming call).       **/
/**     06.  Send a 'B400'X operation (accept call).            **/
/**     07.  Receive '0001'X operation(s) (incoming data) from  **/
/**          the source application program and write it to the **/
/**          file opened in step 1).                            **/
/**     08.  Receive a 'B301'X operation (clear call indication). **/
/**     09.  Send a 'B100'X operation to respond locally to the **/
/**          clearing of the connection.                        **/
/**     10.  Disable the link enabled in step 3).               **/
/**                                                             **/
/**  A data queue will be actively used to manage the operation **/
/**  of this program.  Data queue support will be used to monitor **/
/**  for the completion of the enable and disable routines, as  **/
/**  well as timer expirations and incoming data.  Timers are   **/
/**  used to ensure that there will never be an infinite wait on **/
/**  the data queue.  If a timer expires, the link enabled will **/
/**  be disabled and the program will stop.                     **/
/**                                                             **/
/**                                                             **/
/** Inputs:                                                     **/
/** The program expects the following input parameters:         **/
/**      Line Name:   This is the name of the line description  **/
/**                   that will be used to call the QOLELINK API. **/
/**                   The line must be an X.25 line with at least **/
/**                   one SVC of type *SVCBOTH or *SVCIN.       **/
```

Figure A-2 (Part 1 of 13). C/400 Compiler Listing for the Target Application

```
/**                                                             **/
/**      CommHandle:  This is the logical name that will be used **/
/**                   to identify the link enabled.             **/
/**                                                             **/
/**      Remote DTE Address:  The is the Local Network Address  **/
/**                           of system A.                      **/
/**                                                             **/
/**                                                             **/
/** Outputs:                                                    **/
/** Current status of the file transfer will be provided when   **/
/** running this program. If an error should occur, then a      **/
/** message will be displayed indicating where the error occurred **/
/** and the program will end.        If the program completes   **/
/** successfully, a "successful completion" message will be     **/
/** posted.                                                     **/
/**                                                             **/
/*****************************************************************/

#include "header"

void senddata(sendparms *a, char *b, desc *c, char *d, char *e, int f);

void sndformat1(sendparms *a,char *b, char *c, char *d, qlindparms *e);

void sndformat2 (sendparms *a, char *b, char *c);

void setfilters (hdrparms *a);

void byte (char *a, int b, char *c, int d);

void printespec (espec *a);

void settimer(unsigned short *a,char *b,qentry *c,usrspace *d,char *e);

void dequeue (int a, char *b, qentry *c, usrspace *d);

void putdata (char *a, int b, FILE *c);

void x25lind (qlindparms *a, char *b);

void disablelink (disableparms *a, char *b, usrspace *c);

void handler (disableparms a, usrspace *b);

sigdata_t *sigdata(void);

/*********************************************************/
/***************   Start Main Program   ******************/
/*********************************************************/

main (int argc, char *argv??(??))
{

/************  Variable  Declarations      ******************/

  usrspace inbuff,       /*  Input Data Buffer */
           indesc,       /*  Input Buffer Descriptor  */
           outbuff,      /*  Output Data Buffer  */
           outdesc,      /*  Output Buffer Descriptor */
           qname;        /*  Data Queue  */

  int length,                /* Data Queue key length  */
      inc, i, j;                   /*  counters  */
  unsigned short expctid;      /*  Message ID that is expected */
  char commhandle??(10??),     /*  Command Line Parameter  */
       rmtdte??(17??),         /*  Remote DTE Address  */
       *buffer,                /* Pointer to buffer     */
       key??(256??);           /* Data Queue key identifier  */
  desc *descriptor;            /* Pointer to buffer descriptor  */
/**  definitions for API functions  **/
  enableparms enable;
  disableparms disable;
  sendparms send;
  recvparms recv;
  setfparms setf;
  timerparms timer;
  qlindparms qlind;
  qentry dataq;
  hdrparms *header;
```

Figure A-2 (Part 2 of 13). C/400 Compiler Listing for the Target Application

```
/****** Annnnddddd.... they're off!!  **********/
1

/***--- Open the file to put the received data.     ----**/
if ((fptr = fopen("UDCS_APPLS/OUTFILE))", "w")) == NULL)
  {
   printf("Unable to open target output file in UDCS_APPLS LIB.\n");
   printf("The Program was terminated.\n\n");
   return;
  }
/***--- Open the display file for error handling.   ----**/
if ((screen = fopen("ERRORSPEC", "ab+ type = record")) == NULL)
  {
   printf("Unable to open display file.\n");
   printf("The Program was terminated.\n\n");
   return;
  }

/***--- Set the Exception Handler        ----**/
signal(SIGABRT,&handler);

/** Clear the command line parameters  **/
strncpy(enable.linename, "          ", 10);    /* Clear linename */
strncpy(commhandle, "          ", 10);     /* Clear Commhandle */
strncpy(rmtdte, "                ", 17);     /* Clear Remote DTE */

/** Receive command line Parameters  **/
strncpy(enable.linename, argv??(1??), strlen(argv??(1??)));
strncpy(commhandle, argv??(2??), strlen(argv??(2??)));
strncpy(rmtdte, argv??(3??), strlen(argv??(3??)));
rmtdte??(strlen(argv??(3??))??) = '\0';

/**   Initialize the user spaces  **/
strncpy(inbuff.library, "UDCS_APPLS", 10);    /*  Input Buffer */
strncpy(inbuff.name, "TARGETIBUF", 10);
strncpy(indesc.library, "UDCS_APPLS", 10);    /* Input B Desc  */
strncpy(indesc.name, "TARGETIDSC", 10);
strncpy(outbuff.library, "UDCS_APPLS", 10);   /*  Output Buffer*/
strncpy(outbuff.name, "TARGETOBUF", 10);
strncpy(outdesc.library, "UDCS_APPLS", 10);   /* Output B Desc */
strncpy(outdesc.name, "TARGETODSC", 10);
strncpy(qname.library, "UDCS_APPLS", 10);   /*  Data queue  */
strncpy(qname.name, "X25DTAQ   ", 10);

/*****   retrieve the line description information  ******/
x25lind (&qlind, enable.linename);
if ((qlind.retcode != 0) || (qlind.reason != 0))
   {
   printf("Query line description failed.\n");
   printf("Return code = %d\n", qlind.retcode);
   printf("Reason code = %d\n\n", qlind.reason);
   return;
   }

/*****   Hard Code the QOLELINK Input Parameters  ******/
enable.maxdtax25 = 512;
enable.keylength = 3;
strncpy(enable.keyvalue, "RCV", 3);
2

/**-------  Enable the link  -----------**/
QOLELINK (&(enable.retcode), &(enable.reason), &(enable.tdusize),\
    &(enable.numunits), &(enable.maxdtalan), &(enable.maxdtax25),\
    (char *)&inbuff, (char *)&indesc, (char *)&outbuff,\
    (char *)&outdesc, &(enable.keylength), enable.keyvalue,\
    (char *)&qname, enable.linename, commhandle);

if ((enable.retcode != 0) || (enable.reason != 0))
   {
   printf("Line %.10s with Commhandle %.10s was NOT ENABLED.\n",\
           enable.linename, commhandle);
   printf("Return code = %d\n", enable.retcode);
   printf("Reason code = %d\n\n", enable.reason);
   return;
   }
```

Figure A-2 (Part 3 of 13). C/400 Compiler Listing for the Target Application

```
3

/**-------  Set a timer for Enable link  --------**/
expctid = 0xF0F0;
settimer(&expctid, "Enable", &dataq, &qname, commhandle);
if (expctid != 0xF0F0)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }


/*************************************************************/
/*********----  Set a Filter for the Link  --------*********/
/*************************************************************/
4

QUSPTRUS(&outbuff, &header);   /*  get the output buffer pointer */
header->function = 1;          /*  add a filter   */
header->type = 0;              /*  X.25 PID only  */
header->number = 1;            /*  set 1 filter   */
header->length = 16;           /*  X.25 filter length  */
setfilters(header);            /*  Fill in the filter format */

/*********----  Set the filter for the Link  --------*********/
QOLSETF (&(setf.retcode), &(setf.reason), &(setf.erroffset),\
            commhandle);
if ((setf.retcode != 0) || (setf.reason != 0))
   {
   printf("Set Filters Return Code = %.2d\n", setf.retcode);
   printf("Set Filters Reason Codes = %.4d\n", setf.reason);
   printf("Set Filters Error Offset = %.4d\n", setf.erroffset);
   return;
   }

/*************************************************************/
/****  Receive the incoming call packet and accept the call **/
/*************************************************************/

/**-------  Set a timer to receive data  --------**/
expctid = 0xF0F3;
settimer(&expctid, "Inc Call ", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }
5

/**********  Receive the Incoming Data     **********/
QUSPTRUS (&inbuff, &buffer);
QUSPTRUS (&indesc, &descriptor);

QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
        &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
        &(recv.dataavail), &(recv.errorspecific), commhandle);

if ((recv.retcode != 0) || (recv.reason != 0))
   {
   printf("Recv incoming call packet failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

/*** Interpret the Received Operation  ***/
if (recv.operation != 0xB201)
   {
   printf("Recvd operation %x instead of B201", recv.operation);
   disablelink (&disable, commhandle, &qname);
   return;        /****  End the program  ***/
   }
```

Figure A-2 (Part 4 of 13). C/400 Compiler Listing for the Target Application

6

```
/*****************************************************************/
/** Send a response to accept the call and establish a connection */
/*****************************************************************/

/****   Get pointers to the user spaces.   ******/
QUSPTRUS(&outbuff, &buffer);
QUSPTRUS(&outdesc, &descriptor);

/*******   Set up Send Packet    *********/
send.ucep = 62;                   /*  set UCEP to be 62  */
send.pcep = recv.pcep;            /*  get the PCEP number  */
send.operation = 0xB400;          /*  send a call request response*/
send.numdtaelmnts = 1;            /*  send one data unit   */

/**-----   Send the packet   ----------------**/
sndformat1 (&send, buffer, rmtdte, commhandle, &qlind);
if ((send.retcode != 0) || (send.reason != 0))
   {
   printf("Data NOT sent for commhandle %.9s\n", commhandle);
   printf("Return code = %d\n", send.retcode);
   printf("Reason code = %d\n", send.reason);
   printf("new pcep %d\n\n", send.newpcep);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

printf("An X.25 SVC connection was completed\n\n");
```

7

```
/*********************************************************/
/****    Receive Incoming Data    ***********************/
/*********************************************************/

/**-------  Set a timer to receive data  --------**/
expctid = 0xF0F3;
settimer(&expctid, "Inc Data ", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }

/********---  Receive the Incoming Data     ----******/
/** Get pointer to user space  **/
QUSPTRUS (&inbuff, &buffer);
QUSPTRUS (&indesc, &descriptor);

/**  Receive the data   **/
QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
       &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
       &(recv.dataavail), &(recv.errorspecific), commhandle);

if ((recv.retcode != 0) || (recv.reason != 0))
   {
   printf("Recv op for first data unit failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }
```

Figure A-2 (Part 5 of 13). C/400 Compiler Listing for the Target Application

8

```
/*****************************************************************/
/*******   Start a loop to read in all the incoming data    ***/
/*****************************************************************/

i = 1;
while (recv.operation == 0x0001)
   {
   printf("%d Data Recvd {%.4x}.\n\n", i++, recv.operation);

   /**  Store all the data units in the file  **/
   for (j = 1; j <= recv.numdtaunits; j++) {
      putdata (buffer + (j - 1)*enable.tdusize,\
               descriptor->length, fptr);
      descriptor = (desc *)((char *)descriptor + sizeof(desc));
      }   /* for */

   /**-------  Set a timer to wait for more data  -------**/
   if (recv.dataavail == 0)
      {
      /** Set timer  **/
      expctid = 0xF0F3;
      settimer(&expctid, "Wt Inc Dta", &dataq, &qname, commhandle);
      if (expctid != 0xF0F3)
         {
         disablelink (&disable, commhandle, &qname);
         return;
         }
      }

   /** Get pointer to user space  **/
   QUSPTRUS (&inbuff, &buffer);
   QUSPTRUS (&indesc, &descriptor);

   /**  Receive the data   **/
   QOLRECV (&(recv.retcode), &(recv.reason), &(recv.ucep),\
          &(recv.pcep), &(recv.operation), &(recv.numdtaunits),\
          &(recv.dataavail), &(recv.errorspecific), commhandle);

   }    /**  End  Receive data  while loop   ******/
```

9

```
/*************************************************/
/********** Receive the Clear indication **********/
/*************************************************/

if ((recv.retcode != 83) || (recv.reason != 4002))
   {
   printf("Recv opr for clear request failed\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

/* Interpret the Received Operation  */
if (recv.operation != 0xB301)
   {
   printf("Recvd operation %x instead of B301", recv.operation);
   disablelink (&disable, commhandle, &qname);
   return;         /****  end the program  ***/
   }
```

Figure A-2 (Part 6 of 13). C/400 Compiler Listing for the Target Application

10

```
/*************************************************************/
/***********  Send local response to clear indication ***********/
/*************************************************************/
/****   Get pointers to the user spaces.   ******/
QUSPTRUS(&outbuff, &buffer);
QUSPTRUS(&outdesc, &descriptor);

/*******   Set up the packet         ***************/
send.operation = 0xB100;       /*  send a clear request packet  */
send.numdtaelmnts = 1;         /*  send one data unit  */

/**------   Send the packet   ----------------**/
sndformat2 (&send, buffer, commhandle);

if ((send.retcode != 0) && (send.reason != 0))
   {
   printf("Response not sent for clear connection\n");
   printf("Return code = %d\n", send.retcode);
   printf("Reason code = %d\n", send.reason);
   printf("new pcep %d\n\n", send.newpcep);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

/*****************************************************/
/*********** Receive the Clear Confirmation  **********/
/*****************************************************/

/**-------  Set a timer to receive data  --------**/
expctid = 0xF0F3;
settimer(&expctid, "Clr Cnfrm", &dataq, &qname, commhandle);
if (expctid != 0xF0F3)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }

if ((recv.retcode != 00) || (recv.reason != 0000))
   {
   printf("Recv failed for clear confirmation\n");
   printf("return code %d\n", recv.retcode);
   printf("reason code %d\n", recv.reason);
   printespec(&(send.errorspecific));

   disablelink (&disable, commhandle, &qname);
   return;
   }

/* Interpret the Received Operation  */
if (recv.operation != 0xB101)
   {
   printf("Recvd opr %x instead of opr B301\n", recv.operation);
   disablelink (&disable, commhandle, &qname);
   return;
   }
```

11

```
/*************************************/
/**  disable the link and end program  **/
/*************************************/

disablelink (&disable, commhandle, &qname);

printf("TARGET application completed OK!\n\n");

}    /* End Main */
```

Figure A-2 (Part 7 of 13). C/400 Compiler Listing for the Target Application

```
/*************************************************************/
/************* Start Subroutine Section    *******************/
/*************************************************************/

/*************************************************************/
/************  Routine to fill X.25 Format I   ***********/

void sndformat1 (sendparms *send,
                 char *buffer,
                 char *rmtdte,
                 char *commhandle,
                 qlindparms *qlind)

{
  format1 *output = (format1 *) buffer;
  register int counter;
  register querydata *qd;

  qd = (querydata *)&(qlind->userbuffer);
  output->type = 0;                  /*  not used  */
  output->logchanid = 0x0;
  output->sendpacksize = qd->x25data.defsend;
  output->sendwindsize = qd->x25data.windowsend;
  output->recvpacksize = qd->x25data.defrecv;
  output->recvwindsize = qd->x25data.windowrecv;

  output->dtelength = strlen(rmtdte);               /*  not used  */
  byte(output->dte, 16, rmtdte, strlen(rmtdte));    /*  not used  */
  output->dbit = 0;
  output->cug = 0;                                  /*  not used  */
  output->cugid = 0;                                /*  not used  */
  output->reverse = 0;                              /*  not used  */
  output->fast = 0;                                 /*  not used  */
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->calllength = 0;
  byte(output->callud, 128, "00", 2);
  output->misc??(0??) = 0;
  output->misc??(1??) = 0;
  output->misc??(2??) = 0;
  output->misc??(3??) = 0;
  output->maxasmsize = 16383;
  output->autoflow = 32;

  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));

}  /*  End sndformat1 Subroutine  */
/*************************************************************/
/************  Routine to fill X.25 Format II   ***********/

void sndformat2 (sendparms *send,
                 char *buffer,
                 char *commhandle)

{
  format2 *output = (format2 *) buffer;

  output->type = 1;
  output->cause = 'FF';
  output->diagnostic = 'FF';
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->length = 0;
  byte(output->userdata, 128, "", 0);

  QOLSEND (&(send->retcode), &(send->reason), &(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep),\
           commhandle, &(send->operation), &(send->numdtaelmnts));

}  /*  End sndformat2 Subroutine  */
```

Figure A-2 (Part 8 of 13). C/400 Compiler Listing for the Target Application

```
/*****************************************************************/
/**************   Fill in the Buffer for the Filter  ***************/

void setfilters (hdrparms *header)

{
  x25filter *filters;

  filters = (x25filter *)header->filters;
  filters??(0??).pidlength = 1;
  filters??(0??).pid = 0x21;            /*  set the protocol ID  */
  filters??(0??).dtelength = 0;         /* no DTE used in filter */
  byte(filters??(0??).dte, 12, "", 0);
  filters??(0??).flags = 0x0;
  filters??(0??).flags += 0x80;      /* Set Reverse Charging to no  */
  filters??(0??).flags += 0x40;      /*  Set Fast Select to no  */

}  /*  End setfilters Subroutine  */

/*****************************************/
/********    Routine to disable   ***********/

void disablelink (disableparms *disable,
                  char *commhandle,
                  usrspace *qname)

{
  qentry dataq;
  unsigned short expctid;

  disable->vary = 1;     /*  Hard code device to vary off  */

  /**  Call disable link  **/
  QOLDLINK (&(disable->retcode), &(disable->reason),\
                        commhandle, &(disable->vary));

  if ((disable->retcode != 0) && (disable->reason != 00))
     {
     printf ("Link %.10s did not disabled.\n", commhandle);
     printf ("return code = %d\n", disable->retcode);
     printf ("reason code = %d\n\n", disable->reason);
     }
  else
     printf ("%.10s link disabled \n", commhandle);

  /**-------  Set a timer to receive message  --------**/
  expctid = 0xF0F1;
  settimer(&expctid, "Disable  ", &dataq, qname, commhandle);
  if (expctid != 0xF0F1)
     {
     printf("Disable link did not complete successfully");
     return;
     }

  /**  close the files  **/
  fclose(fptr);
  fclose(screen);

}  /*  End disablelink Subroutine  */


/*****************************************************************/
/**    Routine to convert string to Hexadecimal format   ******/

void byte (char *dest,
           int dlength,
           char *source,
           int slength)
```

Figure A-2 (Part 9 of 13). C/400 Compiler Listing for the Target Application

```
{
  register int counter;
  char holder??(2??);

  for (counter=0;counter<dlength;counter++)
   dest??(counter??)=0;
  for (counter=slength-1;counter>=0;counter--)
   if isxdigit(source??(counter??))
     {
      holder??(0??)=source??(counter??);
      holder??(1??)='\0';
      if  (counter % 2 == 0)
        dest??(counter/2??) += (char) hextoint(holder)*16;
      else dest??(counter/2??) += (char) hextoint(holder);
     }

}  /*  End byte Subroutine  */

/*****************************************************************/
/**    Routine to display the ErrorSpecific output       ******/
/*****************************************************************/

void printespec(espec *errorspecific)

{
  especout outparms;

  QXXFORMAT(screen, "ERRORSPEC ");
  sprintf(outparms.hwecode, "%.8X", errorspecific->hwecode);
  sprintf(outparms.timestamp, "%.8X%.8X", errorspecific->timestamphi,\
                                          errorspecific->timestamplo);
  sprintf(outparms.elogid, "%.8X", errorspecific->elogid);
  if (errorspecific->flags & 0x40)
    outparms.fail = 'Y';
  else outparms.fail = 'N';
  if (errorspecific->flags & 0x20)
    outparms.zerocodes = 'Y';
  else outparms.zerocodes = 'N';
  if (errorspecific->flags & 0x10)
    outparms.qsysopr = 'Y';
  else outparms.qsysopr = 'N';
  sprintf(outparms.cause,"%.2X", errorspecific->cause);
  sprintf(outparms.diagnostic, "%.2X", errorspecific->diagnostic);
  sprintf(outparms.erroffset, "%.6d", errorspecific->erroroffset);
  fwrite(&outparms, 1, sizeof(especout), screen);
  fread("", 0, 0, screen);

}  /*  End printespec Subroutine  */


/*****************************************************************/
/*******   Dequeues the Incoming Message and processes it ******/

void dequeue (int length,
              char *key,
              qentry *dataq,
              usrspace *qname)

{
  char fldlen??(3??),
       waittime??(3??),
       keylen??(2??),
       senderid??(2??),
       *pointer,
       order??(2??);
  register int counter;

  waittime??(0??) = 0;
  waittime??(1??) = 0;
  waittime??(2??) = 0x1D;   /*  Hard code a delay of infinite   */
  keylen??(0??) = 0;
  keylen??(1??) = 0x3F;  /* Hard code a keylength of 3  */
  senderid??(0??) = 0;
  senderid??(1??) = 0x0F;
  strncpy(order, "EQ", 2);
```

Figure A-2 (Part 10 of 13). C/400 Compiler Listing for the Target Application

```
  /*  Clear the data structures  **/
  fflush(stdin);
  pointer = (char *)dataq;
  for (counter = 0; counter < 336; counter++)
    pointer??(counter??) = 0;

  strncpy (dataq->type, "       ", 7);
  while ((strncmp(dataq->type, "*USRDFN", 7) != 0) || (fldlen == 0))
     QRCVDTAQ(qname->name, qname->library, fldlen, dataq, waittime,\
           order, keylen, key, senderid,"");

} /*  End dequeue Subroutine  */

/*********************************************************/
/*************   Set a timer and dequeue next entry  ******/

void settimer (unsigned short *expctid,
               char *process,
               qentry *dataq,
               usrspace *qname,
               char *commhandle)

{
timerparms timer;
disableparms disable;
int length;
char key??(6??);

  timer.interval = 20000;              /*  set timer for 20 seconds */
  timer.establishcount = 1;            /*  set establish count to 1 */
  timer.keylength = 3;                 /*  key value  */
  strncpy(timer.keyvalue, "TGT", 3);   /*  set key value /
  timer.operation = 1;                 /*  set a timer  */

  /* Call QQLTIMER   */
  QQLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
          timer.handlein, (char *)qname, &(timer.operation),\
          &(timer.interval), &(timer.establishcount),\
          &(timer.keylength), timer.keyvalue, timer.userdata);

  if ((timer.retcode != 0) || (timer.reason != 0))
     {
     printf("%s timer failed while being set.\n", process);
     printf("Return code = %d\n", timer.retcode);
     printf("Reason code = %d\n\n", timer.reason);
     }

  /**-------  Dequeue an entry  --------**/
  strncpy(key, "TGT", 3);
  length = 3;
  dequeue (length, key, dataq, qname);

  /*******     Cancel timer   -----***/
  if (dataq->msgid != 0xF0F4)
     {
     strncpy(timer.handlein, timer.handleout, 8);
     timer.operation = 2;           /*  Cancel one timer  */

     QQLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
           timer.handlein, (char *)qname, &(timer.operation),\
           &(timer.interval), &(timer.establishcount),\
           &(timer.keylength), timer.keyvalue, timer.userdata);

     if ((timer.retcode != 0) || (timer.reason != 0))
        {
        printf("%s timer failed while being canceled\n", process);
        printf("Return code = %d\n", timer.retcode);
        printf("Reason code = %d\n\n", timer.reason);
        }
     }
  if (dataq->msgid != *expctid)
     {
     printf ("A %.4X message ID was received instead of %.4X\n",\
              dataq->msgid, *expctid);
     printf ("%s completion message was not received\n", process);
     *expctid = dataq->msgid;
     }

} /*  End settimer Subroutine  */
```

Figure A-2 (Part 11 of 13). C/400 Compiler Listing for the Target Application

```
/*********************************************************/
/**  x25lind:  Read a record into buf and return length   **/

void x25lind (qlindparms *qlind, char *linename)
{
register int counter;

  for(counter=0;counter<256;counter++)
     qlind->userbuffer??(counter??)=0;

  qlind->format = 0x01;
  QQLQLIND (&(qlind->retcode), &(qlind->reason), &(qlind->nbytes),\
           qlind->userbuffer, linename, &(qlind->format));

} /*  End x25lind Subroutine  */

/*********************************************************/
/**  putdata:  Read a record into buf and return length   **/

void putdata (char *buf,
              int dtalen,
              FILE *fptr)

{
int i;

   for (i = 0; i < dtalen; i++)
      fwrite(buf + i, 1, 1, fptr);

} /*  End putdata Subroutine  */


/*  Exception handler, so that a failure will not
    kill the program, and any associated data!!  */

void handler(disableparms disable, usrspace *qname)
{
  sigdata_t *data;

  disablelink(&disable, "*ALL      ", qname);
  printf("The program received an excecption.\n");
  printf("Disable Link was called & the program was terminated.\n\n");

  data=sigdata();
  data->sigact->xhalt=0;
  data->sigact->xrtntosgnler=0;
  data->sigact->xresigprior=0;
  data->sigact->xresigouter=0;

} /*  End handler Subroutine  */

     * * * * *  E N D  O F  S O U R C E  * * * * *


     * * * * *  I N C L U D E S  * * * * *
 Include Name     Last change        Actual Include Name
 header           90/12/19 08:49:31  UDCS_APPLS/QCSRC/HEADER
 typedefs         90/12/19 08:49:31  UDCS_APPLS/QCSRC/TYPEDEFS
 stdio.h          90/11/12 17:24:55  QCC/H/STDIO
 stddef.h         90/11/12 17:24:54  QCC/H/STDDEF
 errno.h          90/11/12 17:24:49  QCC/H/ERRNO
 signal.h         90/11/12 17:24:53  QCC/H/SIGNAL
 ctype.h          90/11/12 17:24:49  QCC/H/CTYPE
 stdarg.h         90/11/12 17:24:54  QCC/H/STDARG
 stdlib.h         90/11/12 17:24:56  QCC/H/STDLIB
 signal.h         90/11/12 17:24:53  QCC/H/SIGNAL
 xxasio.h         90/11/12 17:24:57  QCC/H/XXASIO
 xxcvt.h          90/11/12 17:24:58  QCC/H/XXCVT
 string.h         90/11/12 17:24:56  QCC/H/STRING
 ctype.h          90/11/12 17:24:49  QCC/H/CTYPE
 hexconv          90/12/19 08:49:27  UDCS_APPLS/QCSRC/HEXCONV
 stdio.h          90/11/12 17:24:55  QCC/H/STDIO
 * * * * *  E N D  O F  I N C L U D E S  * * * * *
```

Figure A-2 (Part 12 of 13). C/400 Compiler Listing for the Target Application

```
* * * * *  M E S S A G E  S U M M A R Y  * * * * *
Total  Info   Warning   Error    Severe    Terminal
       (0-4)  (5-19)  (20-29)   (30-39)   (40-99)
  0      0      0        0         0         0
* *  E N D  O F  M E S S A G E  S U M M A R Y  * *


IBM SAA C/400           UDCS_APPLS/TARGET
 ROUTINE          BLOCK NUMBER  SCOPE  TYPE
 <MAIN>                 2       LOCAL  MAIN-PROGRAM
 __sigdata              6       LOCAL  PROCEDURE
 __sgnl                12       LOCAL  PROCEDURE
 __frdinit             49       LOCAL  PROCEDURE
 __getrec              50       LOCAL  PROCEDURE
 __putrec              51       LOCAL  PROCEDURE
 __frdexit             52       LOCAL  PROCEDURE
 __fwrinit             53       LOCAL  PROCEDURE
 __fwrexit             54       LOCAL  PROCEDURE
 QXXFORMAT            129       LOCAL  PROCEDURE
 __strlen             164       LOCAL  PROCEDURE
 __strncmp            168       LOCAL  PROCEDURE
 __strncpy            170       LOCAL  PROCEDURE
 inttohex             181       ENTRY   PROCEDURE
 hextoint             182       ENTRY   PROCEDURE
 sndformat1           197       ENTRY   PROCEDURE
 sndformat2           198       ENTRY   PROCEDURE
 setfilters           199       ENTRY   PROCEDURE
 byte                 200       ENTRY   PROCEDURE
 printespec           201       ENTRY   PROCEDURE
 settimer             202       ENTRY   PROCEDURE
 dequeue              203       ENTRY   PROCEDURE
 putdata              204       ENTRY   PROCEDURE
 x25lind              205       ENTRY   PROCEDURE
 disablelink          206       ENTRY   PROCEDURE
 handler              207       ENTRY   PROCEDURE
 main                 208       ENTRY   PROCEDURE
Program TARGET was created in library UDCS_APPLS.
* * *  E N D  O F  C O M P I L A T I O N  * * *
```

Figure A-2 (Part 13 of 13). C/400 Compiler Listing for the Target Application

The following reference numbers and explanations correspond to the reference numbers in the target application's program listing.

1 Call the C library routines fopen() and signal() to open the target file and set up a signal handler to process AS/400 exceptions, respectively. If an exception situation is encountered, the handler() will be called to perform clean-up in order for the program to end.

2 Call the QOLELINK API to enable the line description using the line name and communications handle passed as input parameters to this program.

3 Call the QOLTIMER API to time the completion of the enable link operation. If the timer expires before the enable-complete message is posted on the this program's data queue, then this program will end.

4 Call the QUSPTRUS API to obtain a pointer to the beginning of the output buffer user space. The output buffer will be used to construct a filter list for the call to the QOLSETF API.

5 Call the QOLRECV API to receive inbound data after reading an incoming data message that was posted on the program's data queue by the user-defined communications support. Since these programs are operating using the communications services of X.25, the first data unit the target program should see is a X'B201' operation signalling an incoming call was received.

6 Call the QOLSEND API with a X'B400' operation to accept the incoming X.25 call. A connection is now established between the source and target application programs.

7 The target program will now set a timer by calling the QOLTIMER API and wait for incoming data. If the timer expires before any incoming data is received, then this program will call the QOLDLINK API, and end.

8 This is the main receive loop for the target program. When data is received from the source program, it will be written to the target file opened during the initialization of this program. The loop will process until a message other than incoming-data entry is read from the program's data queue.

9 Call the QOLSEND API with a X'B001' operation to locally close the connection.

10 Receives a X'B101' operation from the user-defined communications support. This is a local confirmation of X'B100' operation.

11 Call the QOLDLINK API to disable the link previously enabled and end.

Using the Operational Assistant Exit Program for Operational Assistant Backup

The following contains a CL example of a user-written exit program for doing Operational Assistant backup.

```
   PGM    PARM (&PROD &FLAG &OPTIONS &DEVS &TAPSET &RETCODE)
   DCL    VAR(&PRODID) TYPE(*CHAR) LEN(10)     /* Calling product. +
             Will be 'QEZBACKUP' when called from Operational Assistant. */
   DCL    VAR(&FLAG) TYPE(*CHAR) LEN(10)       /* Indicates whether +
                                    before or after backup.     */
   DCL    VAR(&DEVS) TYPE(*CHAR) LEN(40)       /* Devices used.  */
   DCL    VAR(&TAPSET) TYPE(*CHAR) LEN(4)      /* Tape set name  */
   DCL    VAR(&RETCODE) TYPE(*CHAR) LEN(7)     /* Return code    */
   DCL    VAR(&OPTIONS) TYPE(*CHAR) LEN(10)    /* Options used   */
   DCL    VAR(&MSG) TYPE(*CHAR) LEN(512)       /* Message text   */

   IF     COND(&FLAG *EQ '*BEFORE    ') THEN(DO)
   /*-----------------------------------------------------------*/
   /*  Insert commands to be run before the backup here.     */
   /*-----------------------------------------------------------*/
   ENDDO

   IF     COND(&FLAG *EQ '*AFTER     ') THEN(DO)
   /*-----------------------------------------------------------*/
   /*  Insert commands to be run after the backup here.      */
   /*-----------------------------------------------------------*/
   ENDDO
   ENDPGM
```

Creating a Program Temporary Fix Exit Program

This example exit program written in CL, covers the following possible changes in the logical state of the PTF:

- Loaded to temporarily applied
- Temporarily applied to permanently applied
- Temporarily applied to temporarily removed
- Temporarily removed to permanently removed
- Temporarily applied to permanently removed

The example program shows where you can add your code. You can write a PTF exit program in any programming language.

Note: This example does not show the apply-temporary to apply-permanent case. This is because the PTF already called the exit program.

Do not assume the default values for parameters on CL commands or for library lists. Users can change these values. Library lists can vary from one system to another.

```
/****  START OF SPECIFICATIONS  *************************************/
/*                                                                   */
/* PGM NAME: ?                                                       */
/* PGM TYPE: CL program                                              */
/*                                                                   */
/*                                                                   */
/* FUNCTION:                                                         */
/*            THIS EXIT PROGRAM IS CALLED DURING ANY                 */
/*            OF THE FOLLOWING CASES.                                */
/*                                                                   */
/*            APPLY TEMPORARILY -  (user defined)                    */
/*                                                                   */
/*            APPLY PERMANENTLY -  (user defined)                    */
/*                                                                   */
/*            REMOVE TEMPORARILY - (user defined)                    */
/*                                                                   */
/*            REMOVE PERMANENTLY - (user defined)                    */
/*                                                                   */
/* Input:     PARM1 - CHAR(7)  - Product ID                          */
/*            PARM2   CHAR(7)  - PTF ID                              */
/*            PARM3 - CHAR(6)  - Product release                     */
/*            PARM4   CHAR(4)  - Product option ID                   */
/*            PARM5   CHAR(4)  - Product load ID                     */
/*            PARM6   CHAR(10) - PTF library                         */
/*            PARM7   CHAR(50) - User data                           */
/*            PARM8 - CHAR(1)  - Current PTF Status                  */
/*                               0 - LOADED BUT NOT APPLIED          */
/*                               1 - APPLIED TEMPORARILY             */
/*            PARM9   CHAR(1)  - PTF Operation                       */
/*                               0 - REMOVE TEMPORARILY              */
/*                               1 - APPLY TEMPORARILY               */
/*                               2 - APPLY PERMANENTLY               */
/*                               3 - REMOVE PERMANENTLY              */
/*                                                                   */
/*******  END OF SPECIFICATIONS  *************************************/
PGM PARM(&PARM1 &PARM2 &PARM3 &PARM4 &PARM5 &PARM6 &PARM7 &PARM8 &PARM9)
/*-------------------------------------------------------------------*/
/*                                                                   */
/* DECLARE INPUT PARAMETERS                                          */
/*                                                                   */
/*-------------------------------------------------------------------*/

  DCL  &PARM1   TYPE(*CHAR) LEN(7)   /* Product ID                   */
  DCL  &PARM2   TYPE(*CHAR) LEN(7)   /* PTF ID                       */
  DCL  &PARM3   TYPE(*CHAR) LEN(6)   /* Product release              */
  DCL  &PARM4   TYPE(*CHAR) LEN(4)   /* Product option ID            */
  DCL  &PARM5   TYPE(*CHAR) LEN(4)   /* Product load ID              */
  DCL  &PARM6   TYPE(*CHAR) LEN(10)  /* PTF library                  */
  DCL  &PARM7   TYPE(*CHAR) LEN(50)  /* User data                    */
  DCL  &PARM8   TYPE(*CHAR) LEN(1)   /* Current PTF status           */
  DCL  &PARM9   TYPE(*CHAR) LEN(1)   /* PTF operation                */
  /*-------------------------------------------------------------------*/
  /*                                                                   */
  /* DECLARE VARIABLES                                                 */
  /*                                                                   */
  /*-------------------------------------------------------------------*/
  DCL &ACTION   TYPE(*CHAR) LEN(1)   /* PTF action to occur          */
  DCL &STATUS   TYPE(*CHAR) LEN(1)   /* PTF current status           */
                                     /* Handle exceptions            */
  MONMSG     MSGID(CPF0000) EXEC(GOTO CMDLBL(HDLERR))


    CHGVAR VAR(&ACTION) VALUE(&PARM9) /* Gets action being performed  */
    CHGVAR VAR(&STATUS) VALUE(&PARM8) /* Gets current PTF status      */

    /*-----------------------------------------------------------------*/
    /*     THE CURRENT STATUS OF THE PTF IS "LOADED (NOT APPLIED)"     */
    /*-----------------------------------------------------------------*/
    IF (&STATUS = '0') THEN(DO)   /* If PTF is loaded but not applied */
      IF (&ACTION = '1') THEN(DO) /* If action is temporarily         */
                                  /* applied then                     */
          /*?---- TEMP APPLY -  ADD YOUR STATEMENTS HERE -----        */
      ENDDO
      IF (&ACTION = '2') THEN(DO)    /* If action is permanently      */
                                     /* applied then                  */
          /*?---- PERM APPLY - ADD YOUR STATEMENTS HERE -----         */
      ENDDO
    ENDDO                            /* End of loading the PTF       */

    /*-----------------------------------------------------------------*/
    /*     THE CURRENT STATUS OF THE PTF IS "APPLIED TEMPORARILY"      */
    /*-----------------------------------------------------------------*/
    IF (&STATUS = '1') THEN(DO)      /* If PTF is temporarily         */
                                     /* applied then                  */
      IF (&ACTION = '0') THEN(DO)    /* If action is temporarily      */
                                     /* removed then                  */
          /*?---- TEMPORARILY REMOVE - ADD YOUR STATEMENTS HERE ---   */
      ENDDO
      IF (&ACTION = '3') THEN(DO)    /* If action is permanently      */
                                     /*  removed then                 */
          /*?---- PERMANENTLY REMOVE - ADD YOUR STATEMENTS HERE ----- */
      ENDDO
    ENDDO                            /* End of remove the PTF        */

    /*-----------------------------------------------------------------*/
    /*     HANDLE ALL ERROR CONDITIONS                                 */
    /*-----------------------------------------------------------------*/
   HDLERR:
       /* Try to back out any changes already made */
       /* If nothing to back out or back-out operation was successful */
          SNDPGMMSG MSGID(CPF3638) MSGF(QCPFMSG) MSGTYPE(*ESCAPE)
       /* Else permanent changes not backed out   */
          SNDPGMMSG MSGID(CPF3639) MSGF(QCPFMSG) MSGTYPE(*ESCAPE)

ENDPGM                               /* Return to external caller    */
```

Submit Debug Command API Examples

This section contains examples of the Submit Debug Command API. All examples are in the C-programming language. Each example contains a fragment of a program, a debug language statement that appears in the input buffer, and the results produced in the receiver variable.

The null termination symbol (Ø) denotes the end of a character string in the examples that follow.

Break Statement Example

C Program Fragment: Assume program operation is suspended in the program shown in Figure A-3 on page A-54 just before line 6 runs.

```
Line  C Source

 1    #include  stdio.h
 2    int T[] = {1,2,3,5,7,11,13,17,23,29};
 3    int BinarySearch(int v, int f, int l);
 4    main()
 5    { int result;
 6      result = BinarySearch(17,0,9);
 7      printf("result= "); printf("%d",result); printf(" \n");
 8    }
```

Figure A-3. Program for Break Example

Input Buffer

```
BREAK 7 WHEN result > 5
```

Receiver Variable

Offset	Field	Value
0	Bytes returned Bytes available Entry count	58 58 2
12	Result type Break results count Reserved	BreakR 2
24	Result type Line number Reserved	BreakPositionR 1 5
36	Result type Expression text offset Expression text length	ExpressionTextR 48 10
48	String space	Result > 5Ø

Scalar Evaluate Statement Example

C Program Fragment: Consider the C program fragment in Figure A-4. Variable i defines an integer.

```
Line  C Source

 1    int i = 29.
```

Figure A-4. Program for Scaler Evaluate Example

Input Buffer

```
EVAL i
```

Receiver Variable

Offset	Field	Value
0	Bytes returned Bytes available Entry count	64 64 4
12	Result type Evaluation count Reserved	EvaluationR 3
24	Result type Expression text offset Expression text length	ExpressionTextR 60 1
36	Result type Expression value offset Expression value length	ExpressionValueR 62 2
48	Result type Expression type Reserved	ExpressionTypeR Int__32
60	String space	iØ29Ø

Structure Evaluate Statement Example

C Program Fragment: Consider the C program fragment in Figure A-5.

```
Line  C Source

 1    struct {
 2        int i;
 3        float f;
 4        struct {
 5           char c;
 6           enum e {red,yellow};
 7        } s2;
 8    } s1 = { 1 , 5.0, {'a' , red } };
```

Figure A-5. Program for Structure Evaluate Example

Input Buffer

```
EVAL s1
```

Receiver Variable

Offset	Field	Value
0	Bytes returned Bytes available Entry count	247 247 16
12	Result type Evaluation count Reserved	EvaluationR 4
24	Result type Expression text offset Expression text length	ExpressionTextR 204 4
36	Result type Expression value offset Expression value length	ExpressionValueR 209 1
48	Result type Expression type Reserved	ExpressionTypeR Int__32
60	Result type Evaluation count Reserved	EvaluationR 4
72	Result type Expression text offset Expression text length	ExpressionTextR 211 4
84	Result type Expression value offset Expression value length	ExpressionValueR 216 7

Offset	Field	Value
96	Result type Expression type Reserved	ExpressionTypeR Real_32
108	Result type Evaluation count Reserved	EvaluationR 4
120	Result type Expression text offset Expression text length	ExpressionTextR 224 7
132	Result type Expression value offset Expression value length	ExpressionValueR 232 3
144	Result type Expression type Reserved	ExpressionTypeR Char_8
156	Result type Evaluation count Reserved	EvaluationR 4
168	Result type Expression text offset Expression text length	ExpressionTextR 236 7
180	Result type Expression value offset Expression value length	ExpressionValueR 244 3
192	Result type Expression type Reserved	ExpressionTypeR Enum_32
204	String space	See Note.
Note: s1.iØ1Øs1.fØ5.0E+00Øs1.s2.cØ'a'Øs1.s2.eØredØ		

Offset	Field	Value
12	Result type Step Count Reserved	StepR 1

Step Statement Example

C Program Fragment: Assume program operation is suspended in the program shown in Figure A-6 just before line 6 runs.

```
Line  C Source

 1    #include  stdio.h
 2    int T[] = {1,2,3,5,7,11,13,17,23,29};
 3    int BinarySearch(int v, int f, int l);
 4    main()
 5    { int result;
 6      result = BinarySearch(17,0,9);
 7      printf("result= "); printf("%d",result); printf(" \n");
 8    }
```

Figure A-6. Program for Step Example

Input Buffer

STEP

Receiver Variable

Offset	Field	Value
0	Bytes Returned Bytes Available Entry Count	24 24 1

Appendix B. Authority for Call Level Interfaces

The following table lists all the call-level interfaces supported on the system. The public authority listed is the public authority shipped with the system. The public authority for ILE APIs is the authority of the service program or binding directory.

Note: If a field in the table is blank, there is no applicable information for that API.

Figure B-1 (Page 1 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Abnormal End	CEE4ABN	QILE	*USE	
Absolute Function	CEESxABS	QILE	*USE	
Add Commitment Resource	QTNADDCR		*USE	
Add List Entry	QUIADDLE		*USE	
Add List Multiple Entries	QUIADDLM		*USE	
Add Pop-Up Window	QUIADDPW		*USE	
Add Print Application	QUIADDPA		*USE	
Add Product License Information	QLZADDLI		*EXCLUDE	CRTPRDDFN PKGPRDOPT
Add User Index Entries	QUSADDUI		*USE	
AID Spelling	QTWAIDSP		*EXCLUDE	
Arccosine	CEESxACS	QILE	*USE	
Arcsine	CEESxASN	QILE	*USE	
Arctangent	CEESxATN	QILE	*USE	
Arctangent2	CEESxAT2	QILE	*USE	
Backspace on Scroller Line	QsnSclBS	QSNAPI	*USE	
Basic Random Number Generator	CEERANO	QILE	*USE	
Calculate Day-of-Week from Lilian Date	CEEDYWK	QILE	*USE	
Change COBOL Main Program	QLRCHGCM		*CHANGE	
Change Current Job	QWCCCJOB		*USE	
Change Directory Entry Attributes	QHFCHGAT		*USE	
Change Exception Message	QMHCHGEM		*USE	
Change File Pointer	QHFCHGFP		*USE	
Change Library List	QLICHGLL		*USE	CHGLIBL
Change Low-Level Environment	QsnChgEnv	QSNAPI	*USE	
Change Mode Name	QNMCHGMN		*USE	
Change Object Description	QLICOBJD		*USE	
Change Office Program	QOGCHGOE		*EXCLUDE	
Change Pool Attributes	QUSCHGPA		*USE	CHGSBSD CHGSYSVAL
Change Pool Tuning Information	QWCCHGTN		*EXCLUDE	
Change Previous Sign-On Date	QSYCHGPR		*USE	
Change Session	QsnChgSsn	QSNAPI	*USE	
Change User Password	QSYCHGPW		*USE	CHGPWD
Change User Space	QUSCHGUS		*USE	
Change User Space Attributes	QUSCUSAT		*USE	
Change Window	QsnChgWin	QSNAPI	*USE	
Check Command Syntax[1]	QCMDCHK		*USE	
Check Spelling	QTWCHKSP		*EXCLUDE	

Figure B-1 (Page 2 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Check User Authority to an Object	QSYCUSRA		*USE	CHKOBJ
Check User Special Authorities	QSYCUSRS		*USE	
Clear Buffer	QsnClrBuf	QSNAPI	*USE	
Clear Data Queue[1]	QCLRDTAQ		*USE	
Clear Field Table	QsnClrFldTbl	QSNAPI	*USE	
Clear Screen	QsnClrScr	QSNAPI	*USE	
Clear Scroller	QsnClrScl	QSNAPI	*USE	
Clear Window	QsnClrWin	QSNAPI	*USE	
Clear Window Message	QsnClrWinMsg	QSNAPI	*USE	
Close Application	QUICLOA		*USE	
Close Directory	QHFCLODR		*USE	
Close Spooled File	QSPCLOSP		*USE	
Close Stream File	QHFCLOSF		*USE	
Close Virtual Terminal Path	QTVCLOVT		*USE	
Conjugate of Complex	CEESxCJG	QILE	*USE	
Construct a Condition Token	CEENCOD	QILE	*USE	
Control File System	QHFCTLFS		*USE	
Control Office Services	QOCCTLOF		*EXCLUDE	
Control Trace	QWTCTLTR		*USE	
Convert Authority Values to MI Value	QSYCVTA		*USE	
Convert Date and Time Format	QWCCVTDT		*USE	
Convert Date to Lilian Format	CEEDAYS	QILE	*USE	
Convert Edit Code	QECCVTEC		*USE	
Convert Edit Word	QECCVTEW		*USE	
Convert Integers to Seconds	CEEISEC	QILE	*USE	
Convert Lilian Date to Character Format	CEEDATE	QILE	*USE	
Convert Seconds to Character Timestamp	CEEDATM	QILE	*USE	
Convert Seconds to Integers	CEESECI	QILE	*USE	
Convert Sort Sequence Table	QLGCNVSS		*EXCLUDE	
Convert Timestamp to Number of Seconds	CEESECS	QILE	*USE	
Convert Type	QLICVTTP		*USE	
Copy Buffer	QsnCpyBuf	QSNAPI	*USE	
Copy Stream File	QHFCPYSF		*USE	
Cosine	CEESxCOS	QILE	*USE	
Cotangent	CEESxCTN	QILE	*USE	
Create a Session	QsnCrtSsn	QSNAPI	*USE	
Create a Window	QsnCrtWin	QSNAPI	*USE	
Create Command Buffer	QsnCrtCmdBuf	QSNAPI	*USE	
Create Directory	QHFCRTDR		*USE	
Create Heap	CEECRHP	QILE	*USE	
Create Input Buffer	QsnCrtInpBuf	QSNAPI	*USE	
Create Low-Level Environment	QsnCrtEnv	QSNAPI	*USE	
Create Product Definition	QSZCRTPD		*EXCLUDE	
Create Product Load	QSZCRTPL		*EXCLUDE	
Create Program	QPRCRTPG		*EXCLUDE	
Create Program Temporary Fix	QPZCRTFX		*EXCLUDE	

Figure B-1 (Page 3 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Create Spooled File	QSPCRTSP		*EXCLUDE	
Create User Index	QUSCRTUI		*USE	
Create User Queue	QUSCRTUQ		*USE	
Create User Space	QUSCRTUS		*USE	
Decompose a Condition Token	CEEDCOD	QILE	*USE	
Define Heap Allocation Strategy	CEE4DAS	QILE	*USE	
Delete Buffer	QsnDltBuf	QSNAPI	*USE	
Delete Directory	QHFDLTDR		*USE	
Delete Field ID Definition	QsnDltFldId	QSNAPI	*USE	
Delete List	QUIDLTL		*USE	
Delete Low-Level Environment	QsnDltEnv	QSNAPI	*USE	
Delete Product Definition	QSZDLTPD		*EXCLUDE	
Delete Product Load	QSZDLTPL		*EXCLUDE	
Delete Stream File	QHFDLTSF		*USE	
Delete User Index	QUSDLTUI		*USE	DLTUSRIDX
Delete User Queue	QUSDLTUQ		*USE	DLTUSRQ
Delete User Space	QUSDLTUS		*USE	DLTUSRSPC
Deregister Application	QNMDRGAP		*USE	
Deregister APPN Topology Information	QNMDRGTI		*USE	
Deregister File System	QHFDRGFS		*EXCLUDE	
Deregister Filter Notifications	QNMDRGFN		*EXCLUDE	
Disable Link	QOLDLINK		*USE	
Discard Heap	CEEDSHP	QILE	*USE	
Dispatch a Message	CEEMOUT	QILE	*USE	
Display Command Line Window	QUSCMDLN		*USE	CALL QCMD
Display Directory Panels	QOKDSPDP		*USE	
Display Help	QUHDSPH		*USE	
Display Panel	QUIDSPP		*USE	
Display Scroller Bottom	QsnDspSclB	QSNAPI	*USE	
Display Scroller Top	QsnDspSclT	QSNAPI	*USE	
Display Window	QsnDspWin	QSNAPI	*USE	
Dump Flight Recorder	QWTDMPFR		*USE	
Dump Lock Flight Recorder	QWTDMPLF		*EXCLUDE	
Edit	QECEDT		*USE	
Enable Link	QOLELINK		*USE	
End a Window	QsnEndWin	QSNAPI	*USE	
End Application	QNMENDAP		*USE	
End Data Stream Translation Session	QD0ENDTS		*USE	
End Source Debug	QteEndSourceDebug	QTEDBGS	*USE	
Error Function and Its Complement	CEESxERx	QILE	*USE	
Execute Command[1]	QCMDEXC		*USE	
Exponential Base e	CEESxEXP	QILE	*USE	
Exponentiation	CEESxXPx	QILE	*USE	
Factorial	CEE4SIFAC	QILE	*USE	
Filter Problem	QSXFTRPB		*USE	
Find a Control Boundary	CEE4FCB	QILE	*USE	

Figure B-1 (Page 4 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Floating Complex Divide	CEESxDVD	QILE	*USE	
Floating Complex Multiply	CEESxMLT	QILE	*USE	
Force Buffered Data	QHFFRCSF		*USE	
Free Storage	CEEFRST	QILE	*USE	
Gamma Function	CEESxGMA	QILE	*USE	
Generate a Beep	QsnBeep	QSNAPI	*USE	
Generate Alert	QALGENA		*USE	
Generate PTF Name	QPZGENNM		*USE	
Get a Message	CEEMGET	QILE	*USE	
Get AID	QsnGetAID	QSNAPI	*USE	
Get Current Greenwich Mean Time	CEEGMT	QILE	*USE	
Get Current Local Time	CEELOCT	QILE	*USE	
Get Cursor Address	QsnGetCsrAdr	QSNAPI	*USE	
Get Cursor Address with AID	QsnGetCsrAdrAID	QSNAPI	*USE	
Get Dialog Variable	QUIGETV		*USE	
Get Heap Storage	CEEGTST	QILE	*USE	
Get List Entry	QUIGETLE		*USE	
Get List Multiple Entries	QUIGETLM		*USE	
Get Offset from Universal Time Coordinated to Local Time	CEEGMTO	QILE	*USE	
Get Profile Handle	QSYGETPH		*EXCLUDE	
Get Space Status	QLYGETS		*CHANGE	
Get Spooled File Data	QSPGETSP		*USE	
Get Stream File Size	QHFGETSZ		*USE	
Get String Information	CEEGSI	QILE	*USE	
Get Universal Time Coordinated	CEEUTC	QILE	*USE	
Get, Format, and Dispatch a Message	CEEMSG	QILE	*USE	
Go to Next Tab Position in Scroller Line	QsnSclTab	QSNAPI	*USE	
Go to Start of Current Scroller Line	QsnSclCR	QSNAPI	*USE	
Go to Start of Next Scroller Line	QsnSclNL	QSNAPI	*USE	
Handle a Condition	CEE4HC	QILE	*USE	
Hyperbolic Arctangent	CEESxATH	QILE	*USE	
Hyperbolic Cosine	CEESxCSH	QILE	*USE	
Hyperbolic Sine	CEESxSNH	QILE	*USE	
Hyperbolic Tangent	CEESxTNH	QILE	*USE	
Imaginary Part of Complex	CEESxIMG	QILE	*USE	
Initialize Low-Level Environment Description	QsnInzEnvD	QSNAPI	*USE	
Initialize Session Description	QsnInzSsnD	QSNAPI	*USE	
Initialize Window Description	QsnInzWinD	QSNAPI	*USE	
Insert Cursor	QsnInsCsr	QSNAPI	*USE	
List Active Subsystems	QWCLASBS		*USE	WRKSBS
List Authorized Users	QSYLAUTU		*USE	DSPAUTUSR
List Configuration Descriptions	QDCLCFGD		*USE	
List Database Members	QUSLMBR		*USE	DSPFD
List Database Relations	QDBLDBR		*EXCLUDE	DSPDBR
List Fields	QUSLFLD		*USE	DSPFFD

Figure B-1 (Page 5 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
List ILE Program Information	QBNLPGMI		*USE	DSPPGM
List Job	QUSLJOB		*USE	WRKSBSJOB
List Job Log Messages	QMHLJOBL		*USE	DSPJOBLOG
List Job Schedule Entries	QWCLSCDE		*USE	
List Node List Entries	QFVLSTNL		*USE	
List Nonprogram Messages	QMHLSTM		*USE	
List Objects	QUSLOBJ		*USE	DSPOBJD
List Objects Secured by Authorization List	QSYLATLO		*USE	DSPAUTLOBJ
List Objects That Adopt Owner Authority	QSYLOBJP		*USE	DSPPGMADP
List Objects User Is Authorized to or Owns	QSYLOBJA		*USE	DSPUSRPRF
List Performance Data	QPMLPFRD		*USE	
List Record Formats	QUSLRCD		*USE	DSPFD, DSPFFD
List Registered File Systems	QHFLSTFS		*USE	DSPHFS
List Save File	QSRLSAVF		*USE	DSPSAVF
List Service Program Information	QBNLSPGM		*USE	DSPSRVPGM
List Signed-On Users	QEZLSGNU		*EXCLUDE	WRKUSRTOB
List Spooled Files	QUSLSPL		*USE	WRKSPLF, WRKOUTQ
List Subsystem Job Queues	QWDLSJBQ		*USE	DSPSBSD
List Users Authorized to Object	QSYLUSRA		*USE	DSPAUTL DSPOBJAUT
Lock and Unlock Range in Stream File	QHFLULSF		*USE	
Log Gamma Function	CEESxLGM	QILE	*USE	
Log PTF Information	QPZLOGFX		*EXCLUDE	
Log Software Error	QPDLOGER		*USE	
Logarithm Base e	CEESxLOG	QILE	*USE	
Logarithm Base 10	CEESxLG1	QILE	*USE	
Logarithm Base 2	CEESxLG2	QILE	*USE	
Manipulate REXX External Data Queue[1]	QREXQ		*USE	
Manipulate REXX Variables[1]	QREXVAR		*USE	
Map View Position	QteMapViewPosition	QTEDBGS	*USE	
Mark Heap	CEEMKHP	QILE	*USE	
Modular Arithmetic	CEESxMOD	QILE	*USE	
Move Program Messages	QMHMOVPM		*USE	
Move Stream File	QHFMOVSF		*USE	
Move the Resume Cursor to a Current Return Point	CEEMRCR	QILE	*USE	
Move Window	QsnMovWin	QSNAPI	*USE	
Move Window by User	QsnMovWinUsr	QSNAPI	*USE	
Nearest Integer	CEESxNIN	QILE	*USE	
Nearest Whole Number	CEESxNWN	QILE	*USE	
Open Directory	QHFOPNDR		*USE	
Open Display Application	QUIOPNDA		*USE	
Open Print Application	QUIOPNPA		*USE	
Open Spooled File	QSPOPNSP		*USE	
Open Stream File	QHFOPNSF		*USE	

Figure B-1 (Page 6 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Open Virtual Terminal Path	QTVOPNVT		*USE	
Operational Assistant Attention-Key-Handling (group jobs)	QEZMAIN		*USE	
Operational Assistant Attention-Key-Handling (nongroup jobs)	QEZAST		*USE	
Package Product Option	QSZPKGPO		*EXCLUDE	
Pad between Two Screen Addresses	QsnWrtPadAdr	QSNAPI	*USE	
Pad for N Positions	QsnWrtPad	QSNAPI	*USE	
Positive Difference	CEESxDIM	QILE	*USE	
Present the Command Entry Display	QCMD		*USE	
Present the Command Entry Display	QCL		*USE	
Print Panel	QUIPRTP		*USE	
Print Scroller Data	QsnPrtScl	QSNAPI	*USE	
Process Commands	QCAPCMD		*EXCLUDE	
Process Extended Dynamic SQL	QSQPRCED		*EXCLUDE	
Promote Message	QMHPRMM		*USE	
Put Command Buffer	QsnPutBuf	QSNAPI	*USE	
Put Command Buffer and Perform Get	QsnPutGetBuf	QSNAPI	*USE	
Put Dialog Variable	QUIPUTV		*USE	
Put Input Command	QsnPutInpCmd	QSNAPI	*USE	
Put Output Command	QsnPutOutCmd	QSNAPI	*USE	
Put Spooled File Data	QSPPUTSP		*USE	
Put Window Message	QsnPutWinMsg	QSNAPI	*USE	
Query	QQQQRY		*EXCLUDE	OPNQRYF
Query Century	CEEQCEN	QILE	*USE	
Query Color Support	QsnQryColorSup	QSNAPI	*USE	
Query Display Mode Support	QsnQryModSup	QSNAPI	*USE	
Query If Scroller in Line Wrap Mode	QsnQrySclWrp	QSNAPI	*USE	
Query Keyboard Buffering	QWSQRYWS		*USE	
Query Line Description	QOLQLIND		*USE	
Query 5250	QsnQry5250	QSNAPI	*USE	
Read Build Information	QLYRDBI		*CHANGE	
Read Data from Session	QsnReadSsnDta	QSNAPI	*USE	
Read Directory Entries	QHFRDDR		*USE	
Read from Stream File	QHFRDSF		*USE	
Read from Virtual Terminal	QTVRDVT		*USE	
Read Immediate	QsnReadImm	QSNAPI	*USE	
Read Input Fields	QsnReadInp	QSNAPI	*USE	
Read Modified Alternate	QsnReadMDTAlt	QSNAPI	*USE	
Read Modified Fields	QsnReadMDT	QSNAPI	*USE	
Read Modified Immediate Alternate	QsnReadMDTImmAlt	QSNAPI	*USE	
Read Screen	QsnReadScr	QSNAPI	*USE	
Reallocate Storage	CEECZST	QILE	*USE	
Receive Data	QNMRCVDT		*USE	
Receive Data (communications link)	QOLRECV		*USE	
Receive Data Queue[1]	QRCVDTAQ		*USE	

Figure B-1 (Page 7 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Receive Nonprogram Message	QMHRCVM		*USE	
Receive Operation Completion	QNMRCVOC		*USE	
Receive Program Message	QMHRCVPM		*USE	
Register a User-Written Condition Handler	CEEHDLR	QILE	*USE	
Register Activation Group Exit Procedure	CEE4RAGE	QILE	*USE	
Register Application	QNMREGAP		*USE	
Register APPN Tolopogy Information	QNMRGTI		*USE	
Register Call Stack Entry Termination User Exit Procedure	CEERTX	QILE	*USE	
Register Debug View	QteRegisterDebugView	QTEDBGS	*USE	
Register File System	QHFRGFS		*EXCLUDE	
Register Filter Nofifications	QNMRGFN		*EXCLUDE	
Release Heap	CEERLHP	QILE	*USE	
Release License	QLZARLS		*EXCLUDE	
Release Profile Handle	QSYRLSPH		*EXCLUDE	
Remove All Bookmarks from a Course	QEARMVBM		*USE	STREDU
Remove Commitment Resource	QTNRMVCR		*USE	
Remove Debug View	QteRemoveDebugView	QTEDBGS	*USE	
Remove List Entry	QUIRMVLE		*USE	
Remove Nonprogram Messages	QMHRMVM		*USE	
Remove Pop-Up Window	QUIRMVPW		*USE	
Remove Print Application	QUIRMVPA		*USE	
Remove Program Messages	QMHRMVPM		*USE	
Remove User Index Entries	QUSRMVUI		*USE	
Rename Directory	QHFRNMDR		*USE	
Rename Object	QLIRNMO		*USE	RNMOBJ
Rename Stream File	QHFRNMSF		*USE	
Request License	QLZAREQ		*EXCLUDE	
Resend Escape Message	QMHRSNEM		*USE	
Resize Window	QsnRszWin	QSNAPI	*USE	
Resize Window by User	QsnRszWinUsr	QSNAPI	*USE	
Restore Screen	QsnRstScr	QSNAPI	*USE	
Retrieve AID Code on Read	QsnRtvReadAID	QSNAPI	*USE	
Retrieve Alert	QALRTVA		*USE	
Retrieve Buffer Data Length	QsnRtvBufLen	QSNAPI	*USE	
Retrieve Buffer Size	QsnRtvBufSiz	QSNAPI	*USE	
Retrieve COBOL Error Handler	QLRRTVCE		*CHANGE	
Retrieve Command Information	QCDRCMDI		*USE	DSPCMD
Retrieve Commitment Information	QTNRCMTI		*USE	
Retrieve Configuration Status	QDCRCFGS		*USE	
Retrieve Controller Description	QDCRCTLD		*USE	
Retrieve Current Window	QsnRtvCurWin	QSNAPI	*USE	
Retrieve Cursor Address on Read	QsnRtvReadAdr	QSNAPI	*USE	
Retrieve Data Area	QWCRDTAA		*USE	
Retrieve Data Queue Description[1]	QMHQRDQD		*USE	
Retrieve Data Queue Messages[1]	QMHRDQM		*USE	

Figure B-1 (Page 8 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Retrieve Debug Attributes	QteRetrieveDeubgAttributes	QTEDBGS	*USE	
Retrieve Device Description	QDCRDEVD		*USE	
Retrieve Directory Entry Attributes	QHFRTVAT		*USE	
Retrieve Display Mode	QsnRtvMod	QSNAPI	*USE	
Retrieve Field Information	QsnRtvFldInf	QSNAPI	*USE	
Retrieve File Description	QDBRTVFD			
Retrieve ILE Version and Platform ID	CEEGPID	QILE	*USE	
Retrieve Job Description Information	QWDRJOBD		*USE	DSPJOBD
Retrieve Job Information	QUSRJOBI		*USE	DSPJOB WRKACTJOB
Retrieve Job Queue Information	QSPRJOBQ		*USE	
Retrieve Language ID	QLGRTVLI		*EXCLUDE	
Retrieve Length of Data in Input Buffer	QsnRtvDtaLen	QSNAPI	*USE	
Retrieve Length of Field Data in Buffer	QsnRtvFldDtaLen	QSNAPI	*USE	
Retrieve License Information	QLZARTV		*EXCLUDE	
Retrieve Line Description	QDCRLIND		*USE	
Retrieve List Attributes	QUIRTVLA		*USE	
Retrieve Low-Level Environment Description	QsnRtvEnvD	QSNAPI	*USE	
Retrieve Low-Level Environment User Data	QsnRtvEnvDta	QSNAPI	*USE	
Retrieve Low-Level Environment Window Mode	QsnRtvENvWinMod	QSNAPI	*USE	
Retrieve Main Storage	QVTRMSTG		*USE	STRSST
Retrieve Member Description	QUSRMBRD		*USE	DSPFFD
Retrieve Message	QMHRTVM		*USE	
Retrieve Mode Name	QNMRTVMN		*USE	
Retrieve Module Views	QteRetrieveModuleView	QTEDBGS	*USE	
Retrieve Network Attributes	QWCRNETA		*USE	
Retrieve Nonprogram Message Queue Attributes	QMHRMQAT		*USE	
Retrieve Number of Bytes Read from Screen	QsnRtvReadLen	QSNAPI	*USE	
Retrieve Number of Columns to Shift Scroller	QsnRtvSclNumShf	QSNAPI	*USE	
Retrieve Number of Fields Read	QsnRtvFldCnt	QSNAPI	*USE	
Retrieve Number of Rows to Roll Scroller	QsnRtvSclNumRoll	QSNAPI	*USE	
Retrieve Object Description	QUSROBJD		*USE	RTVOBJD
Retrieve Office Program	QOGRTVOE		*EXCLUDE	
Retrieve Operational Descriptor Information	CEEDOD	QILE	*USE	
Retrieve Output Queue Information	QSPROUTQ		*USE	WRKOUTQD
Retrieve Pointer to Data in Input Buffer	QsnRtvDta	QSNAPI	*USE	
Retrieve Pointer to Field Data	QsnRtvFldDta	QSNAPI	*USE	
Retrieve Pointer to User Space	QUSPTRUS		*USE	ALCOBJ
Retrieve Product Information	QSZRTVPR		*EXCLUDE	
Retrieve Program Associated Space	QCLRPGAS		*EXCLUDE	
Retrieve Program Information	QCLRPGMI		*USE	DSPPGM
Retrieve Program Variable	QTERTVPV		*EXCLUDE	
Retrieve PTF Information	QPZRTVFX		*USE	
Retrieve Read Information	QsnRtvReadInf	QSNAPI	*USE	
Retrieve Registered Filters	QNMRFGF		*EXCLUDE	
Retrieve Request Message	QMHRTVRQ		*EXCLUDE	

Figure B-1 (Page 9 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Retrieve Screen Dimensions	QsnRtvScrDim	QSNAPI	*USE	
Retrieve Service Program Information	QBNRSPGM		*USE	DSPSRVPGM
Retrieve Session Data	QsnRtvSsnDta	QSNAPI	*USE	
Retrieve Session Description	QsnRtvSsnD	QSNAPI	*USE	
Retrieve Session Input Line to Command Line	QsnRtvSsnLin	QSNAPI	*USE	
Retrieve Sort Sequence Table	QLGRTVSS		*USE	
Retrieve Spooled File Attributes	QUSRSPLA		*USE	WRKSPLFA
Retrieve Stopped Position	QteRetrieveStoppedPosition	QTEDBGS	*USE	
Retrieve Subsystem Information	QWDRSBSD		*USE	DSPSBSD
Retrieve System Status	QWCRSSTS		*USE	
Retrieve System Values	QWCRSVAL		*USE	
Retrieve User Authority to Object	QSYRUSRA		*USE	
Retrieve User Index Attributes	QUSRUIAT		*USE	
Retrieve User Index Entries	QUSRTVUI		*USE	
Retrieve User Information	QSYRUSRI		*USE	RTVUSRPRF DSPUSRPRF
Retrieve User Space	QUSRTVUS		*USE	
Retrieve User Space Attributes	QUSRUSAT		*USE	
Retrieve View Text	QteRtvViewText	QTEDBGS	*USE	
Retrieve Window Data	QsnRtvWinDta	QSNAPI	*USE	
Retrieve Window Description	QsnRtvWinD	QSNAPI	*USE	
Retrieve Writer Information	QSPRWTRI		*USE	WRKWTR
Return the Relative Invocation Number	CEE4RIN	QILE	*USE	
Return Default Date String for Country	CEEFMDA	QILE	*USE	
Return Default Date/Time String for Country	CEEFMDT	QILE	*USE	
Return Default Time String for Country	CEEFMTM	QILE	*USE	
Roll Down	QsnRollDown	QSNAPI	*USE	
Roll Scroller Down	QsnRollSclDown	QSNAPI	*USE	
Roll Scroller Up	QsnRollSclUp	QSNAPI	*USE	
Roll Up	QsnRollUp	QSNAPI	*USE	
Run a Command[1]	QCMDEXEC		*USE	
Run a Command in System/38 Environment	QCAEXEC		*USE	
Save Information	QEZSAVIN		*USE	
Save Screen	QsnSavScr	QSNAPI	*USE	
Scan for String Pattern[1]	QCLSCAN		*USE	
Scan String for Mixed Data	QLGSCNMX		*EXCLUDE	
Send Alert	QALSNDA		*USE	
Send Break Message	QMHSNDBM		*USE	
Send Data (communications link)	QOLSEND		*USE	
Send Data Queue[1]	QSNDDTAQ		*USE	
Send Error	QNMSNDER		*USE	
Send Message	QEZSNDMG		*USE	SNDMSG SNDBRKMSG
Send Nonprogram Message	QMHSNDM		*USE	
Send Program Message	QMHSNDPM		*USE	
Send Reply	QNMSNDRP		*USE	

Figure B-1 (Page 10 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Send Reply Message	QMHSNDRM		*USE	
Send Request	QNMSNDRQ		*USE	
Send Request for OS/400 Function	QTVSNDRQ		*USE	
Set Century	CEESCEN	QILE	*USE	
Set COBOL Error Handler	QLRSETCE		*CHANGE	
Set Current Window	QsnSetCurWin	QSNAPI	*USE	
Set Cursor Address	QsnSetCsrAdr	QSNAPI	*USE	
Set Error State	QsnSetErr	QSNAPI	*USE	
Set Field	QsnSetFld	QSNAPI	*USE	
Set Filter	QOLSETF		*USE	
Set Keyboard Buffering	QWSSETWS		*USE	
Set List Attributes	QUISETLA		*USE	
Set Lock Flight Recorder	QWTSETLF		*EXCLUDE	
Set Low-Level Environment Window Mode	QsnSetEnvWinMod	QSNAPI	*USE	
Set Output Address	QsnSetOutAdr	QSNAPI	*USE	
Set Profile	QWTSETP		*EXCLUDE	
Set Screen Image	QUISETSC		*USE	
Set Space Status	QLYSETS		*CHANGE	
Set Stream File Size	QHFSETSZ		*USE	
Set Timer	QOLTIMER		*USE	
Set Trace	QWTSETTR		*USE	
Set Window Services Attributes	QsnSetWinAtr	QSNAPI	*USE	
Shift Scroller Left	QsnShfSclL	QSNAPI	*USE	
Shift Scroller Right	QsnShfSclR	QSNAPI	*USE	
Show Programmer Menu[1]	QPGMMENU		*EXCLUDE[2]	STRPGMMNU
Signal a Condition	CEESGL	QILE	*USE	
Signal the Termination-Imminent Condition	CEETREC	QILE	*USE	
Sine	CEESxSIN	QILE	*USE	
Sort	QLGSORT		*USE	
Sort Input/Output	QLGSRTIO		*USE	
Square Root	CEESxSQT	QILE	*USE	
Start a Window	QsnStrWin	QSNAPI	*USE	
Start Application	QNMSTRAP		*USE	
Start Data Stream Translation Session	QD0STRTS		*USE	
Start New Scroller Line at Current Position	QsnSclLF	QSNAPI	*USE	
Start New Scroller Page	QsnSclFF	QSNAPI	*USE	
Start REXX Interpreter to Run REXX Procedure[1]	QREXX		*USE	STRREXPRC
Start Source Debug	QteStartSourceDebug	QTEDBGS	*USE	STRDBG
Store Program Associated Space	QCLSPGAS		*EXCLUDE	
Submit Debug Command	QteSubmitDebugCommand	QTEDBGS	*USE	
Tangent	CEESxTAN	QILE	*USE	
Test for Omitted Argument	CEETSTA	QILE	*USE	
Toggle Line Wrap/Truncate Mode	QsnTglSclWrap	QSNAPI	*USE	
Transfer of Sign	CEESxSGN	QILE	*USE	
Transform AFP to ASCII	QWPZTAFP		*EXCLUDE	

Figure B-1 (Page 11 of 11). Public Authority for Call-Level Interfaces

API Description	API Name	Service Program Name or Binding Directory Name	Default Public Authority	Similar Commands
Translate Character String	QTBXLATE		*USE	
Translate Data Stream	QD0TRNDS		*USE	
Translate Fields[1]	QDCXLATE		*USE	
Truncate Character Data	QLGTRDTA		*EXCLUDE	
Truncation	CEESxINT	QILE	*USE	
Unregister Call Stack Entry Termination User Exit Procedure	CEEUTX	QILE	*USE	
Unregister User Conditional Handler	CEEHDLU	QILE	*USE	
Update List Entry	QUIUPDLE		*USE	
Validate Language ID	QLGVLID		*USE	
Validity Check Command[1]	QCMDCHK		*USE	
Work with Collector	QPMWKCOL		*USE	STRPFRMON
Work with Jobs	QEZBCHJB		*USE	WRKUSRJB
Work with Messages	QEZMSG		*USE	DSPMSG WRKMSG
Work with Printer Output	QEZOUTPT		*USE	WRKSPLF
Work with Problem	QPDWRKPB		*EXCLUDE	
Write Build Information	QLYWRTBI		*CHANGE	
Write Characters to Scroller	QsnWrtSclChr	QSNAPI	*USE	
Write Data	QsnWrtDta	QSNAPI	*USE	
Write Line to Scroller	QsnWrtSclLin	QSNAPI	*USE	
Write Structured Field Major	QsnWrtSFMaj	QSNAPI	*USE	
Write Structured Field Minor	QsnWrtSFMin	QSNAPI	*USE	
Write to Display	QsnWTD	QSNAPI	*USE	
Write to Stream File	QHFWRTSF		*USE	
Write to Virtual Terminal	QTVWRTVT		*USE	
Write Transparent Data	QsnWrtTDta	QSNAPI	*USE	

1 This API is documented in the *CL Programmer's Guide*.
2 When the system is shipped, the QPGMR user profile has *CHANGE authority to this program. Any user whose group profile is QPGMR is able to call this program and access the programmer menu.

Bibliography

This bibliography lists printed information that you need to use the OS/400 APIs, background information for the functions the APIs perform, and other information relevant to specific types of applications. The manuals are grouped in these categories:

- General-purpose manuals
- Programming language manuals
- Communications manuals
- Document library services (DLS) file system manuals

The lists below give the full title and order number of each manual. When these manuals are referred to in text, a shortened version of the title is used.

If you want more information on a topic while you are using this reference, see the *Publications Guide*, GC41-9678, for related AS/400 publications.

General-Purpose Manuals

These books provide general-purpose and background information for the OS/400 licensed program:

- *Advanced Backup and Recovery Guide*, SC41-8079

  The *Advanced Backup and Recovery Guide* provides information about planning a backup and recovery strategy.

- *Basic Security Guide*, SC41-0047

  The *Basic Security Guide* provides general information about OS/400 security and authorities.

- *Central Site Distribution Guide*, SC41-9993

  The *Central Site Distribution Guide* provides information on how to distribute licensed programs, program temporary fixes (PTFs), and application programs to other systems under IBM licensing agreements.

- *Data Description Specifications Reference*, SC41-9620

  The *DDS Reference* provides detailed descriptions of the entries and keywords needed to describe database files (both logical and physical) and certain device files (for displays, printers, and ICF) external to the user's programs.

- *Database Guide*, SC41-9659

  The *Database Guide* describes database concepts.

- *Device Configuration Guide*, SC41-8106

  The *Device Configuration Guide* provides information on how to configure hardware initially and how to change that configuration. Also included is a description of the different keyboard language types. Keyboard language types are specified when using the TELNET function.

- *Guide to Enabling C2 Security*, SC41-0103

  The *Guide to Enabling C2 Security* provides information about planning, installing, setting up, and managing your AS/400 system to meet the requirements for C2 security. C2 is a level of security defined by the United States Department of Defense.

- *Guide to Programming Application and Help Displays*, SC41-0011

  The *Guide to Programming Displays* provides information about:

  - Creating and working with display files
  - Developing online information
  - Creating panel group objects

  This manual also describes the method by which AS/400 users can send a 5250 data stream to a display device (user-defined data streams).

- *Integrated Language Environment* Concepts*, SC09-1524

  The *ILE* Concepts* describes the concepts and terminology of the Integrated Language Environment of the OS/400 operating system.

- *National Language Support Planning Guide*, GC41-9877

  The *National Language Support Planning Guide* provides information needed to evaluate, plan, and use the AS/400 national language support (NLS) and multilingual capabilities.

- *New User's Guide*, SC41-8211

  The *New User's Guide* provides information about how to sign on and off, send and receive messages, respond to keyboard error messages, use function keys, and control and manage jobs. Also included is a description of keyboard differences.

- *Programming: Control Language Programmer's Guide*, SC41-8077

  The *CL Programmer's Guide* discusses OS/400 functions and concepts that are relevant to programming. It includes descriptions of special-purpose APIs not covered in this manual, such as APIs for command syntax checking and data queue management.

- *Programming: Control Language Reference*, SC41-0030

  The *CL Reference* manual provides a description of the AS/400 control language (CL) and its commands. Each command description includes a syntax diagram, parameters, default values, keywords, and an example.

- *Programming: Reference Summary*, SX41-0028

  The *Programming Reference Summary* provides quick reference information when working with the AS/400 system. This manual contains summaries of information such as system values and OS/400 data description specifications (DDS) keywords.

  If you want to create your own translation tables, this manual discusses creating and editing tables for ASCII line mode.

- *Programming: Work Management Guide*, SC41-8078

  The *Work Management Guide* describes work and job management concepts.

- *Security Reference*, SC41-8083

  The *Security Reference* provides technical information about OS/400 security.

- *Service: Diagnostic Aids – Volume 1*, LY44-0597

  The *Diagnostic Aids – Volume 1* provides a list of available object types in hexadecimal format for use with the object APIs.

- *System Concepts*, GC41-9802

  The *System Concepts* provides a general understanding of the concepts related to the overall design and use of the AS/400 system and its operating system. This manual includes general information about AS/400 functions such as user interface, object management, work management, system management, data management, database, communications, environments, OfficeVision/400, PC Support, and system architecture.

- *System Operator's Guide*, SC41-8082

  The *Operator's Guide* provides information on how to respond to error messages and process and manage jobs on the system. Processing jobs includes working with spooled files and finding your printer output.

- *Systems Application Architecture* SystemView* System Manager/400 User's Guide*, SC41-8201

  The *SystemView* System Manager/400 User's Guide* provides information about the commands and functions available when the SAA* SystemView* System Manager/400 licensed program is installed on one or more AS/400 systems in a network.

Programming Language Manuals

You might refer to these programming language manuals while writing applications with the OS/400 APIs:

- *Languages: Pascal Reference*, SC09-1210

  The *Pascal Reference* provides information about using the AS/400 Pascal programming language. It includes information you can refer to while using AS/400 Pascal, such as a detailed description of the program structure, declarations, data types, routines, variables, and statements in Pascal.

- *Languages: Pascal User's Guide*, SC09-1209

  The *Pascal User's Guide* explains how to enter, compile, run, and debug AS/400 Pascal programs.

- *Languages: System C/400 Programming RPQ P10102 User's Guide and Reference*, SC09-1317

  This manual describes how to enter, compile, test, and debug System C/400 PRPQ programs.

- *Languages: Systems Application Architecture* AD/Cycle* COBOL/400* User's Guide*, SC09-1383

  The *COBOL/400* User's Guide* provides information needed to design, write, test, and maintain COBOL/400 programs on the AS/400 system.

- *Languages: Systems Application Architecture* AD/Cycle* RPG/400* Reference*, SC09-1349

  The *RPG/400* Reference* provides information needed to write programs for the AS/400 system using the RPG/400 programming language. This manual describes, position by position, the valid entries for all RPG specification forms, and provides a detailed description of all the operation codes. This manual also contains information on the RPG logic cycle, arrays and tables, editing functions, and indicators.

- *Languages: Systems Application Architecture* AD/Cycle* RPG/400* User's Guide*, SC09-1348

  The *RPG/400* User's Guide* provides information needed to write, test, and maintain RPG/400 programs on the AS/400 system. The manual provides information on data organizations, data formats, file processing, multiple file processing, automatic report function, RPG command statements, testing and debugging functions, application design techniques, problem analysis, and compiler service information. The differences between the System/38 RPG III, System/38 compatible RPG, and RPG/400 are identified.

- *Languages: Systems Application Architecture* C/400* User's Guide*, SC09-1347

  The *C/400* User's Guide* provides information needed to write application programs or develop programs using the C/400 language.

- *Machine Interface Functional Reference*, SC41-8226

  The *MI Functional Reference* is a comprehensive reference to machine interface (MI) instructions.

- *Programming: GDDM Programming Guide*, SC41-0536

  The *GDDM Programming Guide* provides information about using OS/400 graphical data display manager (GDDM) to write graphics application programs.

- *Programming: REXX/400 Programmer's Guide*, SC24-5553,

  The *REXX/400 Programmer's Guide* explains Procedures Language 400/REXX programming concepts and discusses considerations in using this language on the AS/400 system. It also describes REXX APIs and provides examples that you can use to learn Procedures Language 400/REXX.

- *Programming: REXX/400 Reference*, SC24-5552

  The *REXX/400 Reference* provides an overview of the Procedures Language 400/REXX concepts and includes information about keyword instruction syntax, function syntax, numerics, arithmetic, conditions, input and output streams, testing, and double-byte character set (DBCS) support. The manual also describes REXX APIs.

- *SAA* AD/Cycle* Application Development Manager/400 Introduction and Planning Guide*, GC09-1377

  The *Application Development Manager/400 Introduction and Planning Guide* provides a general discussion of the following:

  - What the Application Development Manager/400 product does, who its users are, and the advantages it provides
  - The relationships between this product and the AS/400 programs and the AD/Cycle* framework
  - The concepts relating to the tasks that application developers and project administrators perform.

- *SAA* AD/Cycle* Application Development Manager/400 User's Guide*, SC09-1376

  The *Application Development Manager/400 User's Guide* provides information needed for Application Development Manager/400 project administrators to define project hierarchies for application development, and to define the creation and movement of parts in a project hierarchy by application developers.

- *Systems Application Architecture* Structured Query Language/400 Programmer's Guide*, SC41-9609

  The *SQL/400* Programmer's Guide* provides an overview of how to design, write, run, and test SQL/400 statements. It also describes the interactive Structured Query Language (SQL).

- *Systems Application Architecture* Structured Query Language/400 Reference*, SC41-9608

  The *SQL/400* Reference* provides information about SQL/400 statements and their parameters.

Communications Manuals

These manuals provide background information for the communications APIs described in this book:

- *Communications and Systems Management Guide (Alerts and Distributed Systems Node Executive)*, SC41-9661

  The *Alerts and DSNX Guide* provides background information for the alert APIs.

- *Communications: Local Area Network Guide*, SC41-0004

  The *Local Area Network Guide* provides information about using an AS/400 system in a token-ring network, Ethernet network, or in a bridged environment.

- *Communications: Management Guide*, SC41-0024

  The *Communications Management Guide* provides information about working with communications status, errors, performance, line speed, and storage requirements.

- *Communications: Operating System/400* Communications Configuration Reference*, SC41-0001

  The *OS/400* Communications Configuration Reference* provides general communications configuration information about lines, controllers, devices, modes, class-of-service, configuration lists, network interfaces, and connection lists.

- *Communications: SNA Upline Facility Programmer's Guide*, SC41-9594

  The *SNA Upline Facility Programmer's Guide* provides information on display data streams using formatted buffers.

- *Communications: X.25 Network Guide*, SC41-0005

  The *X.25 Network Guide* provides information about using an AS/400 system in an X.25 packet-switched network.

- *Communications: 3270 Device Emulation Guide*, SC41-9602

  The *3270 Device Emulation Guide* provides a list of 3270 data stream commands, orders, and attributes that are supported.

The following IBM manuals are not specific to the AS/400 system, but do contain helpful communications information:

- *Advanced Function Printing: Data Stream Reference*, S544-3202, defines the AFP data stream used for advanced function page printing. It describes the use and syntax of data stream components and explains how the structured fields are interpreted in presenting composite pages of text, image, graphic, and bar code data. It specifies the approved content of the AFP data stream to be supported by IBM Print Services in each of the system environments. The AFP data stream is an architected presentation function set of the Mixed Object:Document Content Architecture, which is part of IBM's Systems Application Architecture.

- *IBM 8209 LAN Bridge Customer Information*, SA21-9994

  This *IBM 8209 LAN Bridge Customer Information* manual describes how to set up and use the 8209 Local Area Network Bridge.

- *Intelligent Printer Data Stream Reference*, S544-3417, describes the functions and services associated with IPDS. It is intended for systems programmers and other developers who need such information to develop or adapt a product or program to attach to an IPDS printer in an SAA communications network.

- *Print Services Facility User's Programming Guide for VM*, S544-3512, includes valid table reference codes and how these codes relate to the CHARS attribute and page definitions used in the spooled file APIs.

- *Print Services Facility/MVS Application Programming Guide*, S544-3084, includes valid table reference codes and how these codes relate to the CHARS attribute and page definitions used in the spooled file APIs.

- *Systems Network Architecture Formats*, GA27-3136

  The *SNA Formats* describes the contents of OS/400 alerts in detail.

- *Token-Ring Network Architecture Reference*, SC30-3374
- *Token-Ring Network Problem Determination Guide Kit*, SC30-3374
- *3270 Information Display System: 3274 Control Unit Description and Programmer's Guide*, GA23-0061
- *5250 Functions Reference Manual*, SA21-9247

The following manuals are not produced by IBM and are not specific to the AS/400 system, but do contain helpful communications information:

- *American National Standards Institute/Institute of Electrical and Electronics Engineers 802.2, 1985 - Logical Link Control, International Organization for Standardization/Draft International Standard 8802/2.*
- *American National Standards Institute/Institute of Electrical and Electronics Engineers 802.3, 1985 - Carrier Sense Multiple Access with Collision Detection, International Organization for Standardization/Draft International Standard 8802/3.*
- *American National Standards Institute/Institute of Electrical and Electronics Engineers 802.3a, b, d, c, 1988 -Supplements to Carrier Sense Multiple Access with Collision Detection American National Standards Institute/Institute of Electrical and Electronics Engineers Standard 802.3*, 1985.
- *American National Standards Institute/Institute of Electrical and Electronics Engineers 802.5, 1985 - Token Passing Ring, International Organization for Standardization/Draft International Standard 8802/5.*
- *The International Telegraph and Telephone Consultative Committee, Blue Book, Volume VIII - Fascicle VIII.2, Data Communications Networks: Services and Facilities, Recommendations X.1 - X.32, IXth Plenary Assembly*, Melbourne, November 14-25, 1988.

DLS File System Manuals

Refer to these manuals for detailed information about the document library services (DLS) file system:

- *Document Interchange Architecture: Interchange Document Profile Reference*, SC23-0764

  The *IDP Reference* provides detailed information about the interchange document profile (IDP).

- *Office Services Concepts and Programmer's Guide*, SC41-9758

  The *Office Services Concepts and Programmer's Guide* provides background information for DLS file system users.

- *Systems Application Architecture* OfficeVision/400*: Managing OfficeVision/400*, SC41-9627

  The *Managing OfficeVision/400** describes administration procedures relevant to the DLS file system.

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Readers' Comments—We'd Like to Hear from You!

Application System/400
System Programmer's
Interface Reference
Version 2

Publication No. SC41-8223-02

Overall, how would you rate this manual?

	Very Satisfied	Satisfied	Dissatis-fied	Very Dissatis-fied
Overall satisfaction				

How satisfied are you that the information in this manual is:

Accurate				
Complete				
Easy to find				
Easy to understand				
Well organized				
Applicable to your tasks				
THANK YOU!				

Please tell us how we can improve this manual:

May we contact you to discuss your responses? __ Yes __ No

Phone: (____) _____ Fax: (____) _____

To return this form:

- Mail it
- Fax it
  United States and Canada: **800+937-3430**
  Other countries: **(+1)+507+253-5192**
- Hand it to your IBM representative.

Note that IBM may use or distribute the responses to this form without obligation.

Name

Address

Company or Organization

Phone No.

Readers' Comments—We'd Like to Hear from You!
SC41-8223-02

IBM®

Cut o
Alon

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DEPT 245
IBM CORPORATION
3605 HWY 52 N
ROCHESTER MN 55901-9986

Fold and Tape | **Please do not staple** | Fold and Tape

Cut o
Alon

Readers' Comments—We'd Like to Hear from You!

Application System/400
System Programmer's
Interface Reference
Version 2

Publication No. SC41-8223-02

Overall, how would you rate this manual?

	Very Satisfied	Satisfied	Dissatis-fied	Very Dissatis-fied
Overall satisfaction				

How satisfied are you that the information in this manual is:

Accurate				
Complete				
Easy to find				
Easy to understand				
Well organized				
Applicable to your tasks				
THANK YOU!				

Please tell us how we can improve this manual:

May we contact you to discuss your responses? __ Yes __ No

Phone: (____) _____ Fax: (____) _____

To return this form:

- Mail it
- Fax it
  United States and Canada: **800+937-3430**
  Other countries: **(+1)+507+253-5192**
- Hand it to your IBM representative.

Note that IBM may use or distribute the responses to this form without obligation.

Name

Address

Company or Organization

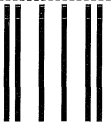
Phone No.

Readers' Comments—We'd Like to Hear from You!
SC41-8223-02

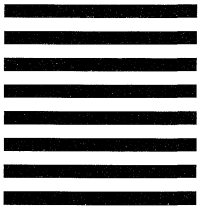
IBM®

Cut
Alon

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DEPT 245
IBM CORPORATION
3605 HWY 52 N
ROCHESTER MN 55901-9986

Fold and Tape | **Please do not staple** | Fold and Tape

Cut o
Along

Readers' Comments—We'd Like to Hear from You!

Application System/400
System Programmer's
Interface Reference
Version 2

Publication No. SC41-8223-02

Overall, how would you rate this manual?

	Very Satisfied	Satisfied	Dissatis-fied	Very Dissatis-fied
Overall satisfaction				

How satisfied are you that the information in this manual is:

Accurate				
Complete				
Easy to find				
Easy to understand				
Well organized				
Applicable to your tasks				
THANK YOU!				

Please tell us how we can improve this manual:

May we contact you to discuss your responses? __ Yes __ No

Phone: (____) _____ Fax: (____) _____

To return this form:

- Mail it
- Fax it
  United States and Canada: **800+937-3430**
  Other countries: **(+1)+507+253-5192**
- Hand it to your IBM representative.

Note that IBM may use or distribute the responses to this form without obligation.

Name _____ Address _____

Company or Organization _____

Phone No. _____

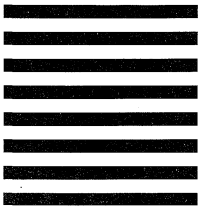
Readers' Comments—We'd Like to Hear from You!
SC41-8223-02

IBM®

Cut c
Along

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DEPT 245
IBM CORPORATION
3605 HWY 52 N
ROCHESTER MN 55901-9986

Fold and Tape | **Please do not staple** | Fold and Tape

Cut o
Along

Program Number: 5738-SS1

Printed in Denmark by Bonde's

SC41-8223-02