IBM i

Programming
API overview and concepts

7.1

IBM i

Programming
API overview and concepts

7.1

Note

Before using this information and the product it supports, read the information in "Notices," on page 573.

This edition applies to IBM i 7.1 (product number 5770-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

Contents

Application programming interfaces

IBM® i application programming interfaces (APIs) allow your application program written in a high-level language to use specific data or functions of the i operating system.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

APIs overview

This API information describes most of the IBM i APIs and some APIs for related licensed programs that run on the i operating system.

Who should use APIs

APIs are intended for experienced application programmers to develop system-level and other IBM i applications. The API information provides reference only. It is neither an introduction to IBM i nor a guide to writing IBM i applications.

How the API information is organized

The API information can be found either by the API name through the API finder or by category through the API categories page.

In the API finder, you can search for APIs by category, by API name, by descriptive name, or by part of the name. You can also search for new APIs, changed APIs, and exit programs.

The API categories are major functional categories, such as backup and recovery, objects, and work management. Within the individual categories, the APIs are organized in alphabetical order as follows:

- By the spelled-out name for the program-based APIs, the service-program-based APIs, and the Integrated Language Environment® (ILE) Common Execution Environment (CEE) APIs.
- By the function name for the UNIX-type APIs.

Compatibility with future releases

IBM intends that the APIs will continue to work as they originally worked, and any existing applications that use the APIs will continue to work without any incompatible changes. Significant architectural changes, however, might necessitate incompatible changes.

Additionally, some API definitions, such as the UNIX type of API definitions, are established by industry standards organizations, where the degree of compatibility is determined by the organizations.

In future releases, IBM also intends that one of the following statements is true:

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

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

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

System APIs or CL commands--when to use each

An API is designed to be used as a programming interface, and a CL command is intended to be entered either interactively or in a CL program.

Before system APIs were offered on the system, you had to either code separate CL programs to perform the needed functions using the appropriate CL commands or code a call to the Execute Command (QCMDEXC) API in your program. Both methods made coding an application on the system more cumbersome (less straightforward and not as fast as possible).

CL commands will always be needed; they are ideal for the interactive user and for CL applications that are performing basic tasks. They provide a complete set of functions on the system.

APIs are not provided as a replacement for CL commands, although in many cases there might be both an API and a CL command that perform the same function. If a CL command and an API provide the same function, at times the API provides more flexibility and information.

Some APIs have no equivalent CL commands. These APIs have been provided in areas where customers and business partners have indicated that they need high-level language (HLL) access.

Actions and system functions of APIs

An API can be categorized by the type of action it performs and by the system function that it relates to.

Listed here are some of the types of APIs that perform actions; several examples of these APIs are discussed in more detail in later topics.

- List APIs, which return lists of information about something on the system.
- Retrieve APIs, which return information to the application program.
- Create, change, and delete APIs, which work with objects of a specified type on the system.
- Other APIs, which perform a variety of actions on the system.

Although many APIs are used alone, some APIs can be used together to perform tasks or functions as in these examples:

- Defining, creating, distributing, and maintaining your own software products.
- Controlling systems and networks, which includes configuration, spooled files, network management, and problem management.
- Handling objects, which includes creating, changing, copying, deleting, moving, and renaming objects on the system.

Related reference:

"Examples: Using data queues or user queues" on page 282
Both data queues and user queues provide a means for one or more processes to communicate asynchronously. Both queues can be processed by first-in first-out (FIFO), last-in first-out (LIFO), or by key.

What's new for IBM i 7.1

Read about new or significantly changed information for the Application programming interfaces topic collection.

You can find a list of new APIs or changed APIs for this release using the API finder.

In addition, the following categories of APIs have been enhanced this release.

- PowerHA® for i APIs

  A sub-category of PowerHA for i APIs has been added in the Cluster category.
- Sockets-related User Exit Programs

  A new subcategory of exit programs has been added in the Unix-type category, Sockets sub-category.

What's new as of October 2015

Miscellaneous technical updates have been made to several APIs.

What's new as of February 2013

Several APIs have been updated to support new SSL protocols and cipher suites.

GSKit API updates have been made to support several new security attributes.

Miscellaneous minor technical updates were also made to a few APIs.

What's new as of October 2012

The new Change Processor Multitasking Information (QWCCHGPR) and Retrieve Processor Multitasking Information (QWCRTVPR) APIs have been added to the Work Management category.

Miscellaneous minor technical updates were also made to a few APIs.

What's new as of 24 April 2012

The new Suspend System Exit Programs and the Resume System Exit Programs exit points for partition migration and hibernation have been added to the Work Management category.

The Command Analyzer Retrieve Exit Program added a new "Before or after indicator" to specify when an exit program should be called (either before or after the command processing program has run.)

Miscellaneous technical updates were made to a number of other APIs.

What's new as of 12 October 2011

The new Compress Files and Directories and the Decompress Files and Directories APIs have been added to the UNIX-Type category under the Integrated File System APIs sub-category.

The new Configure Activation Engine API was added to the Configuration category.

API updates have been made to support Ethernet Link Aggregation and Ethernet Bridging.

Miscellaneous technical updates were made to a number of other APIs.

What's new as of 12 April 2011

Miscellaneous minor technical updates have been made.

What's new as of 10 September 2010

The new Retrieve Workload Capping Group Information API was added to the Software Product category.

Miscellaneous technical updates were made to a number of other APIs.

What's new as of 30 June 2010

Miscellaneous minor technical updates have been made to a number of APIs.

How to see what's new or changed

To help you see where technical changes have been made, the information center uses:

- The » image to mark where new or changed information begins.
- The « image to mark where new or changed information ends.

In PDF files, you might see revision bars (|) in the left margin of new and changed information.

To find other information about what's new or changed this release, see the Memo to users.

PDF file for APIs

You can view and print a PDF file of this information.

To view or download the PDF version of overview and concept information for APIs, select API overview and concepts (about 4500 KB). PDFs for API descriptions are no longer provided.

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF link in your browser.
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html) .

API concepts

An *application programming interface (API)* is a functional interface supplied by the operating system or a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

Some APIs provide the same functions as control language (CL) commands and output file support. Some APIs provide functions that CL commands do not. Most APIs work more quickly and use less system overhead than the CL commands.

API use has the following advantages:

- APIs provide better performance when getting system information or when using system functions that are provided by CL commands or output file support.
- APIs provide system information and functions that are not available through CL commands.
- You can use calls from high-level languages to APIs.
- You can access system functions at a lower level than what was initially provided on the system.
- Data is often easier to work with when returned by an API.

API terminology

Before using the IBM i APIs, you need to understand several terms that refer to IBM i objects.

The system-recognized identifiers are shown in parentheses.

Note: Each term does not apply to every API.

binding directory (*BNDDIR)
An object that contains a list of names of modules and service programs.

data queue (*DTAQ)
An object that is used to communicate and store data used by several programs in a job or between jobs.

module (*MODULE)
An object that is made up of the output of the compiler.

program (*PGM)
A sequence of instructions that a computer can interpret and run. A program can contain one or more modules.

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

user index (*USRIDX)
An object that provides a specific order for byte data according to the value of the data.

user queue (*USRQ)
An object consisting of a list of messages that communicate information to other application programs. Only programming languages that can use machine interface (MI) instructions can access *USRQ objects.

user space (*USRSPC)
An object consisting of a collection of bytes used for storing any user-defined information.

Generic library names

These special values refer to IBM i libraries. You can often use them in API calls in place of specific library names.

***ALL** All libraries, including the QSYS library.

***ALLUSR**
All user-defined libraries with names that do not begin with the letter Q. Although the following libraries with names that begin with the letter Q are provided by IBM, they typically contain user data that changes frequently. Therefore, these libraries are also considered user libraries.

QDSNX QGPL QGPL38 QMGTC QMGTC2 QMPGDATA QMQMDATA QMQMPROC QPFRDATA QRCL QRCL*xxxxx*	QSRVAGT QSYS2 QSYS2*xxxxx* QS36F QUSER38 QUSRADSM QUSRBRM QUSRDIRCF QUSRDIRCL QUSRDIRDB QUSRIJS QUSRINFSKR	QUSRNOTES QUSROND QUSRPOSGS QUSRPOSSA QUSRPYMSVR QUSRRDARS QUSRSYS QUSRVI QUSRVxRxMx

*ALLUSR excludes System/36 libraries that have names starting with the symbol # and that do not contain user data. The following table lists those libraries.

#CGULIB #COBLIB #DFULIB #DSULIB	#RPGLIB #SDALIB #SEULIB	

xxxxx is the number of a primary auxiliary storage pool (ASP).

A different library name, in the format QUSRV*x*R*x*M*x*, can be created by the user for each previous release supported by IBM to contain any user commands to be compiled in a CL program for the previous release. For the QUSRV*x*R*x*M*x* user library, V*x*R*x*M*x* is the version, release, and modification level of a previous release that IBM continues to support.

For more information about QUSRV*x*R*x*M*x* libraries or the *ALLUSR special value, see Special values for the SAVLIB command.

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

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

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

Related reference:

Special values for the SAVLIB command

API naming conventions

Program-based APIs and service-program-based APIs follow similar naming conventions.

Except for the APIs that are defined by formal standards organizations (for example, UNIX-type APIs), an API name starts with the letter Q, followed by 2, 3, or 4 letters that comprise an internal component identifier. The last part of the API name identifies the action or function of the API.

The following table contains all of the verbs that are either part of an API name or are implied verbs associated with an API name.

Table 1. Verbs and abbreviations for program-based and service-program-based APIs

Verb	**Abbreviation**
access	access
Add	ADD, Add
Change	C, CHG, Chg, ch

Table 1. Verbs and abbreviations for program-based and service-program-based APIs (continued)

Verb	Abbreviation
Check	C, CHK, CHECK
Clear	CLR, Clr
Close	CLO, close
Complete	Cmp
Control	CTL
Convert	CVT, CVRT, Convert
Copy	CPY, Cpy
Create	CRT, Crt, create
Customize	CST
Delete	DLT, Dlt
Deregister	DRG, Deregister
Disable	D
Display	DSP, Dsp
Dump	DMP, Dump
duplicate	dup
Edit	EDT
Enable	E
End	END, End
Execute (run)	EXC, EXEC
Filter	FTR
Force	FRC
Generate	GEN
Get (fetch)	G, GET, Get, get
Initialize	Inz
Insert	Ins
link	link
List	L, LST, List
Lock/unlock	LUL
make	mk
Map	Map
Maximize	Mxz
Move	MOV, Mov
Open	OPN, open
Pad	Pad
Print	PRT, Prt
Put	PUT, Put
PutGet	PutGet
Query	Q, QRY, Qry
Read	RD, Read, read
Receive	R, RCV, RECV

Table 1. Verbs and abbreviations for program-based and service-program-based APIs (continued)

Verb	Abbreviation
Register	RG, REG, R, Register
Release	RLS
Remove	RMV, Rmv, Remove, rm
Rename	RNM, rename
Report	Report
Resend	RSN
Reserve	Reserve
Restore	RST, Rst, Restore
reset	rewind
Resize	Rsz
Retrieve	R, RTV, Rtv, Retrieve
Roll	Roll
Save	SAV, Sav, Save
Scan for	SCAN
Send	S, SND, SEND, Send
Set	SET, Set
Shift	Shf
Start	Start, STR, Str
Submit	Submit
Switch	Set
Test	T
Toggle	Tgl
Transform	T
Translate	TR, TRN, XLATE
truncate	truncate
Unregister	U
Update	UPD
Validate	V
Work with	WK, WRK, Wrk
Write	WRT, Wrt, write, W

Related concepts:

"Types of APIs" on page 10
IBM i APIs exist in several operating environments on a system.

Language selection considerations

You can directly use APIs, other than service-program-based APIs, with all the languages that are available with the IBM i operating system.

ILE APIs that are implemented as service programs (*SRVPGM) can be directly accessed only by ILE languages. For non-ILE languages, the Call Service Program Procedure (QZRUCLSP) API is available to indirectly access service-program-based APIs. In some cases, an ILE API also provides a program (*PGM) interface so that non-ILE languages can access the function.

Some APIs also require that particular data types and particular parameter passing conventions be used. The following table shows the languages that are available with the IBM i operating system and the data types that they provide.

Table 2. Language selection considerations–data types

Language	Pointers	Binary 2	Binary 4	Character	Zoned decimal	Packed decimal	Floating point	Structures	Single array	Exception handling
BASIC (PRPQ 5799-FPK)		X	X	X	X [1]	X [1]	X		X	X
ILE C	X	X	X	X		X [6]	X	X	X	X
VisualAge® C++ for IBM i	X	X	X	X		X [7]	X	X	X	X
CL	X	X	X	X		X		X	X [2]	X
ILE CL	X	X	X	X		X		X	X [2]	X
COBOL	X	X	X	X	X	X		X	X	X [3]
ILE COBOL	X	X	X	X	X	X	X	X	X	X [3]
MI	X	X	X	X	X	X	X	X	X	X
Pascal (PRPQ 5799-FRJ)	X	X	X	X	X [4]	X [4]	X	X	X	X
PL/I (PRPQ 5799-FPJ)	X	X	X	X	X	X	X	X	X	X
REXX				X				X [2]	X [2]	X
RPG		X	X	X	X	X		X	X	X [5]
ILE RPG	X	X	X	X	X	X	X	X	X	X [5]

Notes:

1. Refer to the CNVRT$ intrinsic function.
2. There is no direct support, but you can use the substring capability to simulate structures and arrays.
3. COBOL and ILE COBOL programs cannot monitor for specific messages, but these programs can define an error handler to run when a program ends because of an error.
4. There is no direct support, but you can use extended program model (EPM) conversion routines to convert to and from zoned and packed decimal.
5. RPG programs cannot monitor for specific messages, but these programs can define an error handler to run when a program ends because of an error.
6. Packed decimal is implemented in ILE C with the decimal() data type.
7. Packed decimal is implemented in VisualAge C++ for IBM i with the Binary Coded Decimal (BCD) class. The BCD class is the C++ implementation of the C-language's decimal(). The BCD object can be used in API calls because it is binary compatible with the decimal() data type.

The following table shows the languages that are available with the IBM i operating system and the parameter support that they provide. See the reference information for the specific programming language that you plan to use.

Table 3. Language selection considerations–call conventions

Language	Function return values [1]	Pass by reference	Pass by value
BASIC		X	
ILE C	X	X	X
VisualAge C++ for IBM i	X	X	X
CL		X	
ILE CL	X[2]	X	X[2]
COBOL		X	[3]
ILE COBOL	X	X	X

Table 3. Language selection considerations–call conventions (continued)

Language	Function return values [1]	Pass by reference	Pass by value
MI		X	X
Pascal		X	
PL/I		X	
REXX		X	
RPG		X	
ILE RPG	X	X	X

Notes:

1. Return values are used by the UNIX-type APIs and the Dynamic Screen Manager (DSM) APIs.
2. This support is available only when using the Call Bound Procedure (CALLPRC) command.
3. COBOL provides a by-content phrase, but it does not have the same semantics as ILE C pass-by-value.

Related concepts:

"User spaces for list APIs" on page 79
List APIs require a user space to contain returned information.

Types of APIs

IBM i APIs exist in several operating environments on a system.

APIs for the program-based environment

Program-based APIs are called as programs (*PGMs). They are the initial APIs on the system.

Program-based APIs use the following naming conventions:

- Names of program APIs start with the letter Q.
- Names of program APIs are followed by a 2-, 3-, or 4-letter internal component identifier.
- Names of program APIs are limited to 8 characters.
- Names of program APIs must be uppercase.

Related concepts:

"API information format" on page 47
The format of the IBM i API information includes sections such as parameters, authorities and locks, required parameter group, format, field descriptions, and error messages.

"Examples: Program-based APIs" on page 120
These examples demonstrate the use of program-based APIs in several different high-level language programs.

APIs for the service-program-based environment

APIs based on service programs are called as procedures exported from ILE service programs (*SRVPGM).

Bindable procedure APIs, exported from ILE service programs (*SRVPGM), are independent of the high-level languages. This can be useful when mixed languages are involved.

Here are some of the functions provided by the service-program-based APIs:

- Dynamic screen management (DSM)
- National language support
- Mail server framework
- Problem management
- Programming and control language (CL)

- Registration facility
- Source debugger

APIs based on service programs that are *defined by IBM* use the following naming conventions:
- Start with the letter Q.
- Are followed by a 2-, 3-, or 4-character internal component identifier.
- Can be up to 30 characters.
- Are case sensitive.

APIs based on service programs that are *defined by industry standards* follow the industry naming conventions.

ILE service programs (*SRVPGM) use the following naming conventions:
- Start with the letter Q.
- Are followed by a 2-, 3-, or 4-character internal component identifier.
- Are limited to 8 characters.
- Are uppercase.

Bindable APIs are contained within service programs to which the calling program binds. Some bindable APIs provide a program interface for the original program model (OPM) languages. You can usually distinguish between the *SRVPGM interface and the *PGM interface by the name of the API. For example, the registration facility APIs provide both a program and a service program entry point (procedure) interface. For the Register Exit Point API, the service program entry point interface is named QusRegisterExitPoint and the program interface is named QUSRGPT. A bindable procedure name can be up to 30 characters and mixed uppercase and lowercase. A program interface name can be up to 8 characters and is all uppercase.

A binding directory is used for APIs that are contained in service programs. A *binding directory* is a list of names of modules and service programs that provides a reference by name and type. Service programs that export bindable APIs are in the QUSAPIBD binding directory. This binding directory is implicitly used by ILE compilers to resolve the bindable API references; therefore, it is not necessary to explicitly name the service program or the API binding directory when you create programs that use bindable APIs. If you provide your own APIs with the same name, make sure that you also provide your own binding directory or service program.

Most APIs (program-based and service-program-based) have a header file supplied by the IBM i operating system. These header files reside in the optionally installable library QSYSINC. The header files provide the prototypes for the API and define any structures that are used by the API. The QSYSINC library is used by the ILE C compiler to search for header files; therefore, it is not necessary to specify a library qualifier for any header files that reside in the QSYSINC library. When you code in ILE C, remember to enclose the header file name in less-than (<) and greater-than (>) symbols because this affects how the library list is processed in locating the header file.

It is typical for an API that is not retrieving information not to return any output to the caller other than the error code parameter. If no errors occur when APIs are used, the requested function is completed successfully.

Related concepts:

"API information format" on page 47
The format of the IBM i API information includes sections such as parameters, authorities and locks, required parameter group, format, field descriptions, and error messages.

"Examples: Service-program-based APIs" on page 185
These program examples demonstrate the use of service-program-based APIs in several different high-level language programs. The example APIs represent two general functions of APIs: change and

retrieve.

APIs for the ILE Common Execution Environment

The service-program-based APIs with names that begin with CEE are based on an IBM cross-platform language environment specification. The Common Execution Environment (CEE) APIs are intended to be consistent across the IBM systems.

The CEE APIs with names that begin with CEE4 or CEES4 are specific to business computing systems.

The CEE APIs provide the following functions:

- Activation group and control flow management
- Condition management
- Date and time manipulation
- Math functions
- Message services
- Program or procedure call management and operational descriptor access
- Storage management

Related reference:

ILE CEE APIs

Differences between program-based APIs and service-program-based APIs

Program-based APIs and service-program-based APIs are different in API names, parameters, error conditions, and pointer use.

APIs based on service programs include the UNIX-type APIs and the ILE Common Execution Environment (CEE) APIs. You must have the ILE language compiler on your system to develop applications that use any bindable APIs.

The following table shows the differences between program-based APIs and service-program-based APIs.

Table 4. Comparison of program-based and service-program-based APIs

Description	Program APIs	Bindable APIs
API name	Maximum number of characters: 8. Not case sensitive.	Maximum number of characters: 30. Case sensitive.
Required parameters[1]	Displayed in the parameter box.	Displayed in the parameter box.
Optional parameters	Can include optional parameters. The optional parameters form a group. You must either include or exclude the entire group.	No optional parameters.
Omitted parameters	No omitted parameters.	Can include omitted parameters. When these parameters are omitted, you must pass a null pointer.
Error conditions[2]	The error code parameter is common to most of the program-based APIs. It is used to return error codes and exception data to the application. The errors that are returned for a given API are in the form of an error message and a 7-character message identifier.	The error code parameter is common to most of the bindable APIs whose names start with the letter Q. The ILE CEE APIs use feedback codes and conditions. The UNIX-type APIs generally use errnos to provide error-related information.

Table 4. Comparison of program-based and service-program-based APIs (continued)

Description	Program APIs	Bindable APIs
Pointers	Can be used, but are used less frequently than with the bindable APIs.	Because of the greater availability of pointer support in ILE languages, there is a much greater use of pointers in the bindable APIs. The use of pointers can provide a performance advantage.

Notes:

1. The UNIX-type APIs include parameters in a syntax box.
2. Error conditions
   - The UNIX-type APIs use *errnos* and return values.
   - The Dynamic Screen Manager (DSM) supports returned values in addition to the error code parameter.

   The *errnos* are provided as include files in the QSYSINC library.

In the examples that follow, a program-based API and a service-program-based API are used to perform similar functions (log or report software errors). The bindable API uses pointers while the program API does not. Both APIs log software errors by using first-failure data capture (FFDC).

Related concepts:

"API parameters" on page 52
After you decide which API to use, you need to code a call to the API and pass to the API the set of parameters that are appropriate for it.

Related reference:

Errno Values for UNIX-Type Functions

Example in ILE C: Logging software errors (program API without pointers):

This ILE C program calls the Log Software Error (QPDLOGER) API to perform first-failure data capture (FFDC) without using pointers.

The ILE C program physically moves the data that is pointed to, as shown at (1), which slows down performance.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*                                                                   */
/*Program Name:  FFDCPGM1                                            */
/*                                                                   */
/*Program Language:  ILE C                                           */
/*                                                                   */
/*Description:  This program illustrates how to use APIs to log      */
/*              software errors using FFDC.                          */
/*                                                                   */
/*                                                                   */
/*Header Files Included:  except                                     */
/*                        stdio                                      */
/*                        string                                     */
/*                        qmhchgem                                   */
/*                        qpdloger                                   */
/*                        qusec                                      */
/*                                                                   */
/*APIs Used:     QPDLOGER                                            */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
```

```
/*********************************************************************/
/*                        System Includes                            */
/*********************************************************************/
#include <except.h>               /* from QSYSINC/H                  */
#include <stdio.h>                /* from QSYSINC/H                  */
#include <string.h>               /* from QSYSINC/H                  */

/*********************************************************************/
/*                    Miscellaneous Includes                         */
/*********************************************************************/
#include <qmhchgem.h>
#include <qpdloger.h>
#include <qusec.h>

/*********************************************************************/
/*                           Structures                              */
/*********************************************************************/
typedef struct {
    void  *parm1;
    void  *parm2;
    char  *pgm_name;
    int    pgm_name_size;
} ffdc_info_t;

/*********************************************************************/
/*                           Prototypes                              */
/*********************************************************************/
void UNEXPECTED_HDLR(_INTRPT_Hndlr_Parms_T *);

/*********************************************************************/
/*   FUNCTION NAME:  main                                            */
/*                                                                   */
/*   FUNCTION:       Generates exception and then passes control     */
/*                   to exception handler.                           */
/*                                                                   */
/*   INPUT:          Two character strings.                          */
/*                                                                   */
/*   OUTPUT:         NONE                                            */
/*                                                                   */
/*   EXCEPTIONS:     CPFxxxx - All unexpected CPF exceptions         */
/*                   MCHxxxx - All unexpected MCH exceptions         */
/*                                                                   */
/*********************************************************************/
void main(int argc, char *argv[])
{

  /*******************************************************************/
  /* NOTE:  argv will contain the parameters passed in to this       */
  /*        function.  In this case, two parameters are passed       */
  /*        in.                                                      */

  /*******************************************************************/
  /*******************************************************************/
  /* The argv parameter contains the parameters that were passed as  */
  /* character arrays.  argv[0] contains the program name, and the   */
  /* parameter(s) starts with argv[1].                               */
  /*******************************************************************/

  char *nulptr;                   /* Pointer used to generate error  */
  char pgm_name[30];              /* Program name                    */
  volatile ffdc_info_t ffdc_info; /* FFDC info for unexpected error  */

  /*******************************************************************/
  /* Set up FFDC information for unexpected error.                   */
  /*******************************************************************/
  ffdc_info.parm1 = argv[1];
  ffdc_info.parm2 = argv[2];
```

```
  ffdc_info.pgm_name = pgm_name;
  memcpy(pgm_name, argv[0], strlen(argv[0]));
  ffdc_info.pgm_name_size = strlen(argv[0]);

  /*******************************************************************/
  /* Enable the exception handler, and pass ffdc_info into the       */
  /* exception handler via the communications area so that data      */
  /* can be used for FFDC.                                           */
  /*******************************************************************/

#pragma exception_handler (UNEXPECTED_HDLR, ffdc_info, 0, _C2_MH_ESCAPE)

  /*******************************************************************/
  /* Set the pointer to null, then try to increment.  This will      */
  /* generate an MCH3601 error that will be trapped by the           */
  /* unexpected handler.                                             */
  /*******************************************************************/
  nulptr = NULL;
  nulptr++;

#pragma disable_handler

} /* main */

/*********************************************************************/
/*   FUNCTION NAME:  UNEXPECTED_HDLR                                 */
/*                                                                   */
/*   FUNCTION:       Handle unexpected exception.  This exception    */
/*                   handler is used to log the software error via   */
/*                   FFDC.                                           */
/*                                                                   */
/*   INPUT:          Interrupt handler information                   */
/*                                                                   */
/*   OUTPUT:         NONE                                            */
/*                                                                   */
/*   EXCEPTIONS:     CPFxxxx - All unexpected CPF exceptions         */
/*                   MCHxxxx - All unexpected MCH exceptions         */
/*                                                                   */
/*********************************************************************/
void UNEXPECTED_HDLR(_INTRPT_Hndlr_Parms_T *errmsg)
{

  typedef struct {
         char   obj_name[30];
         char   obj_lib[30];
         char   obj_type[10];
  } obj_info_t;

  typedef struct {
         int    data_offset;
         int    data_length;
  } data_info_t;

  char           pgm_suspected[10],
                 msg_id[12],
                 msg_key[4],
                 print_job_log,
                 data[2*(sizeof(char *))],
                *data_item,
                 ile_mod_name[11];
  int            point_of_failure,
                 num_items,
                 num_objs;
  data_info_t    data_info[2];
  obj_info_t     obj_info[1];
  ffdc_info_t   *ffdc_info;
  Qus_EC_t       ErrorCode;
```

```
ErrorCode.Bytes_Provided = 0;

/*****************************************************************/
/* Getting pointer in local storage to the Communications Area.  */
/*****************************************************************/
ffdc_info = (ffdc_info_t *)(errmsg->Com_Area);

/*****************************************************************/
/* Need to notify message handler that we will handle the error. */
/* Leave the message in the job log, just mark it handled.       */
/*****************************************************************/
QMHCHGEM(&(errmsg->Target),            /* Invocation pointer     */
          0,                           /* Call stack counter     */
          (char *)&errmsg->Msg_Ref_Key,/* Message key            */
          "*HANDLE   ",                /* Modification option    */
          "",                          /* Reply text             */
          0,                           /* Reply text length      */
          &ErrorCode);                 /* Error code             */

/*****************************************************************/
/* Set up the suspected program.                                 */
/*****************************************************************/
memcpy(pgm_suspected, "*PRV      ", 10);

/*****************************************************************/
/* Set up the detection identifier.                              */
/*****************************************************************/
memset(msg_id, ' ', 12);
memcpy(msg_id, errmsg->Msg_Id, 7);

/*****************************************************************/
/* Set up the message key.                                       */
/*****************************************************************/
memcpy(msg_key, (char *)&errmsg->Msg_Ref_Key, 4);

/*****************************************************************/
/* Set up point of failure.  Since this example program is small */
/* and we know where the error occurred, we will just put a dummy*/
/* value in.  However, this can be very useful information in    */
/* larger programs.                                              */
/*****************************************************************/
point_of_failure = 100;

/*****************************************************************/
/* Set up to print the job log.                                  */
/*****************************************************************/
print_job_log = 'Y';

/*****************************************************************/
/* Set up data items.                                            */
/*****************************************************************/
data_item = data;

/*****************************************************************/
/* Put in first parameter.                                       */
/*****************************************************************/
memcpy(data_item, (char *)ffdc_info->parm1, sizeof(char *));        (1)

/*****************************************************************/
/* Add in the second parameter.                                  */
/*****************************************************************/
data_item += sizeof(char *);
memcpy(data_item, (char *)ffdc_info->parm2, sizeof(char *));

/*****************************************************************/
/* Reset the data item pointer.                                  */
```

```
  /*****************************************************************/
  data_item -= sizeof(char *);

  /*****************************************************************/
  /* Set up data item offset/length information.                   */
  /*****************************************************************/
  data_info[0].data_offset = 0;
  data_info[0].data_length = sizeof(char *);
  data_info[1].data_offset = sizeof(char *);
  data_info[1].data_length = sizeof(char *);

  /*****************************************************************/
  /* Set up the number of data items.  In this case we only have one.*/
  /*****************************************************************/
  num_items = 2;

  /*****************************************************************/
  /* Set up the object name array.  In this case, we have no objects */
  /* to dump, but we will put dummy values in to illustrate.         */
  /*****************************************************************/
  memcpy(obj_info[0].obj_name, "OBJUSRSPC                     ", 30);
  memcpy(obj_info[0].obj_lib, "QTEMP                         ", 30);
  memcpy(obj_info[0].obj_type, "*USRSPC   ", 10);

  /*****************************************************************/
  /* Set the number of objects in name array.                      */
  /*****************************************************************/
  num_objs = 0;

  /*****************************************************************/
  /* Set up the ILE module name.                                   */
  /*****************************************************************/
  memcpy(ile_mod_name, ffdc_info->pgm_name, ffdc_info->pgm_name_size);

  /*****************************************************************/
  /* Call QPDLOGER to perform FFDC.                                */
  /*****************************************************************/
  ErrorCode.Bytes_Provided = sizeof(ErrorCode);
  QPDLOGER(pgm_suspected,
           msg_id,
           msg_key,
           point_of_failure,
           &print_job_log,
           data_item,
           data_info,
           num_items,
           obj_info,
           num_objs,
           &ErrorCode,
           ile_mod_name);

} /* UNEXPECTED_HDLR */
```

Related reference:

"Example in OPM COBOL: Logging software errors (program API without pointers)" on page 18
One OPM COBOL program registers an error handler. After successful completion of the registration, this program creates a data decimal error. The error causes the error handler, the other OPM COBOL program, to call the Log Software Error (QPDLOGER) API without using pointers.

"Example in OPM RPG: Logging software errors (program API without pointers)" on page 21
This OPM RPG program performs a divide-by-zero operation to cause an exception. The exception is caught with RPG *PSSR support, which calls the Log Software Error (QPDLOGER) API to perform FFDC without using pointers.

"Example in ILE RPG: Logging software error (program API without pointers)" on page 24
This ILE RPG program performs a divide-by-zero operation to cause an exception. The exception is caught with RPG *PSSR support, which calls the Log Software Error (QPDLOGER) API to perform FFDC

without using pointers.

Example in OPM COBOL: Logging software errors (program API without pointers):

One OPM COBOL program registers an error handler. After successful completion of the registration, this program creates a data decimal error. The error causes the error handler, the other OPM COBOL program, to call the Log Software Error (QPDLOGER) API without using pointers.

The program CBLERR1 registers the error handler and causes the error. The program ERRHDL1 receives control and calls the QPDLOGER API to perform FFDC.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

CBLERR1 program

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*  Program:      Register an OPM COBOL Error Handler
*                Cause a data decimal exception to demonstrate
*                   logging of software errors
*
*  Language:     COBOL
*
*  Description:  This program registers an OPM COBOL Error
*                Handler.  After the successful completion of
*                the registration of the error handler, this     ,
*                program creates a data decimal error. This
*                exception causes the error handler to be
*                called which then logs the software error.
*
*  APIs Used:    QLRSETCE - Set COBOL Error Handler
*
*****************************************************************
*
*****************************************************************
 PROGRAM-ID. CBLERR1.
 INPUT-OUTPUT SECTION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Miscellaneous elements
*
 01  MISC.
     05  Y               PIC S9(09) VALUE 0.
     05  ERROR-HANDLER   PIC  X(20) VALUE "ERRHDL1   *LIBL     ".
     05  SCOPE           PIC  X(01) VALUE "C".
     05  ERROR-HANDLER-LIBRARY PIC X(10).
     05  PRIOR-ERROR-HANDLER   PIC X(20).
 01  NUMERIC-GROUP.
     05  X               PIC  9(03).
*
* Beginning of mainline
```

```
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Register the COBOL Error Handler.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
* API for the parameter.
*
     MOVE 16 TO BYTES-PROVIDED.
*
*
* Call the API to register the exit point.
*
     CALL "QLRSETCE" USING ERROR-HANDLER OF MISC,
                           SCOPE OF MISC,
                           ERROR-HANDLER-LIBRARY OF MISC,
                           PRIOR-ERROR-HANDLER OF MISC,
                           QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                            DISPLAY "Error setting handler",
                            STOP RUN.
*
* If the call to register an error handler is successful, then
* cause a the data decimal error (X is initialized to blanks).
*
     ADD X TO Y.
*
* Should not get here due to data decimal error
*
     STOP RUN.
*
* End of MAINLINE
*
```

ERRHDL1 program

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*  Program:      Log a software error
*
*  Language:     COBOL
*
*  Description:  This program receives control for exceptions
*                within a COBOL run unit.  This program is used
*                in conjunction with CBLERR1.                    ,
*                Any exception causes this error handler to be
*                called which then logs the software error.
*
*  APIs Used:    QPDLOGER - Log Software Error
*
*****************************************************************
*
*****************************************************************
 PROGRAM-ID. ERRHDL1.
```

```
 INPUT-OUTPUT SECTION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Miscellaneous elements
*
 01  MISC.
     05  LOG-EXCEPTION-ID PIC X(12).
     05  MESSAGE-KEY      PIC  X(04).
     05  POINT-OF-FAILURE PIC S9(09) BINARY VALUE 1.
     05  PRINT-JOBLOG     PIC  X(01) VALUE "Y".
     05  NBR-OF-ENTRIES   PIC S9(09) BINARY.
     05  NBR-OF-OBJECTS   PIC S9(09) BINARY VALUE 1.
 01  MESSAGE-INFO.
     05  MSG-OFFSET       PIC S9(09) BINARY.
     05  MSG-LENGTH       PIC S9(09) BINARY.
 01  OBJECT-LIST.
     05  OBJECT-NAME      PIC  X(30).
     05  LIBRARY-NAME     PIC  X(30).
     05  OBJECT-TYPE      PIC  X(10) VALUE "*PGM      ".
 LINKAGE SECTION.
 01  CBL-EXCEPTION-ID     PIC  X(07).
 01  VALID-RESPONSES      PIC  X(06).
 01  PGM-IN-ERROR.
     05  PGM-NAME         PIC  X(10).
     05  LIB-NAME         PIC  X(10).
 01  SYS-EXCEPTION-ID     PIC  X(07).
 01  MESSAGE-TEXT         PIC  X(01).
 01  MESSAGE-LENGTH       PIC S9(09) BINARY.
 01  SYS-OPTION           PIC  X(01).
*
* Beginning of mainline
*
 PROCEDURE DIVISION USING CBL-EXCEPTION-ID,
                          VALID-RESPONSES,
                          PGM-IN-ERROR,
                          SYS-EXCEPTION-ID,
                          MESSAGE-TEXT,
                          MESSAGE-LENGTH,
                          SYS-OPTION.
 MAIN-LINE.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
* API for the parameter.
*
     MOVE 16 TO BYTES-PROVIDED.
*
* Record the COBOL Exception id
*
     MOVE SYS-EXCEPTION-ID TO LOG-EXCEPTION-ID.
*
* Record the length of the message replacement data (if any)
*
```

```
     IF MESSAGE-LENGTH > 0
        MOVE 1 TO MSG-OFFSET,
        MOVE MESSAGE-LENGTH TO MSG-LENGTH,
        MOVE 1 TO NBR-OF-ENTRIES,
     ELSE
        MOVE 0 TO MSG-OFFSET,
        MOVE 0 TO MSG-LENGTH,
        MOVE 0 TO NBR-OF-ENTRIES.
*
* For illustration purposes, dump the program object
*
     MOVE PGM-NAME TO OBJECT-NAME.            (1)
     MOVE LIB-NAME TO LIBRARY-NAME.
*
* Call the API to log the software error.
*
     CALL "QPDLOGER" USING PGM-NAME,
                           LOG-EXCEPTION-ID,
                           MESSAGE-KEY,
                           POINT-OF-FAILURE,
                           PRINT-JOBLOG,
                           MESSAGE-TEXT,
                           MESSAGE-INFO,
                           NBR-OF-ENTRIES,
                           OBJECT-LIST,
                           NBR-OF-OBJECTS,
                           QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                            DISPLAY "Cannot log erro".
*
* End the current run unit
*
     MOVE "C" TO SYS-OPTION.
     STOP RUN.
*
* End of MAINLINE
*
```

Related reference:

"Example in ILE C: Logging software errors (program API without pointers)" on page 13
This ILE C program calls the Log Software Error (QPDLOGER) API to perform first-failure data capture (FFDC) without using pointers.

Example in OPM RPG: Logging software errors (program API without pointers):

This OPM RPG program performs a divide-by-zero operation to cause an exception. The exception is caught with RPG *PSSR support, which calls the Log Software Error (QPDLOGER) API to perform FFDC without using pointers.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*************************************************************
F*
F* Program:        Demonstrate use of OPM-based Log Software Error
F*
F* Language:       OPM RPG
F*
F* Description:    This program performs a divide-by-0 operation
F*                 to cause an exception.  This exception is
```

```
F*                 caught using RPG *PSSR support,
F*                 and the exception is then logged as a
F*                 software error.
F*
F* APIs used:      QPDLOGER
F*
F*****************************************************************
E*
E* Arrays used to extract source line number where error happened
E*
E                    SRC         8  1
E                    TGT         8  1
I*
I* Error Code parameter include.  As this sample program uses
I* /COPY to include the error code structure, only the first
I* 16 bytes of the error code structure are available.  If the
I* application program needs to access the variable length
I* exception data for the error, the developer should physically
I* copy the QSYSINC include and modify the copied include to
I* define additional storage for the exception data.
I*
I/COPY QSYSINC/QRPGSRC,QUSEC
I*
I* Define Program Status Data Structure
I*
IPSDS       SDS
I                                       1  10 PGMNAM
I                                      11  150STATUS
I                                      21  28 SRC
I                                      40  46 EXCPID
I                                      81  90 LIBNAM
I*
I* Some miscellaneous fields
I*
IMISC       DS
I                                    B   1   40FAILPT
I                                    B   5   80DATA#
I                                    B   9  120OBJS#
I                                       13  20 TGT
I                                       13  200LIN#C
I*
I* DATA represents the data items to report as part of problem
I*
IDATA       DS                        4096
I*
I* DATAPT defines (via offset and length values) how to read DATA
I*
IDATAPT     DS                         256
I                                    B   1   40DTAOFF
I                                    B   5   80DTALEN
I*
I* OBJS represents the list of objects to spool as part of problem
I*
IOBJS       DS                        2590
I                                       1  30 OBJ1N
I                                      31  60 OBJ1L
I                                      61  70 OBJ1T
C*
C* Prepare for divide-by-zero situation
C*
C                    Z-ADD10       FACT1   50
C                    Z-ADD0        FACT2   50
C*
C* and divide by 0
C*
C           FACT1    DIV  FACT2    RESULT  50
C*
```

```
C* should not get here due to divide-by-0 exception
C*
C                     MOVE '1'       *INLR
C                     RETRN
C*
C* Program exception subroutine:
C*
C           *PSSR     BEGSR
C*
C* Make sure we are not catching an exception due to the *PSSR
C* subroutine itself
C*
C           SWITCH    IFEQ ' '
C                     MOVE '1'       SWITCH  1
C*
C* Set API error code to work in nonexception mode
C*
C                     Z-ADD16        QUSBNB
C*
C* Record the source listing line number that caused the failure
C*
C*      First, extract the numeric portion of the PSDS line number
C*
C                     Z-ADD8         X       10
C                     Z-ADD8         Y       10
C                     Z-ADD0         LIN#C
C           SRC,X     DOWEQ' '
C                     SUB  1         X
C                     END
C           X         DOWGT0
C                     MOVE SRC,X     TGT,Y
C                     SUB  1         X
C                     SUB  1         Y
C                     END
C*
C*      Then record it:
C*
C                     Z-ADDLIN#C     FAILPT
C*
C* Record the status code for the failure
C*
C                     MOVELSTATUS    DATA
C*
C* Record where to find the status data within DATA
C*
C                     Z-ADD0         DTAOFF
C                     Z-ADD5         DTALEN
C                     Z-ADD1         DATA#
C*
C* For illustration purposes also dump the program object as
C* part of logging the software error
C*
C                     MOVELPGMNAM    OBJ1N                (1)
C                     MOVELLIBNAM    OBJ1L
C                     MOVEL'*PGM'    OBJ1T
C                     Z-ADD1         OBJS#
C*
C* Call the Log Software Error API
C*
C                     CALL 'QPDLOGER'
C                     PARM           PGMNAM
C                     PARM EXCPID    MSGID  12
C                     PARM           MSGKEY  4
C                     PARM           FAILPT
C                     PARM 'Y'       JOBLOG  1
C                     PARM           DATA
C                     PARM           DATAPT
```

```
C                     PARM           DATA#
C                     PARM           OBJS
C                     PARM           OBJS#
C                     PARM           QUSBN
C*
C* If an error on the API call, then indicate a terminal error
C*
C           QUSBNC    IFGT 0
C           'TERM ERR'DSPLY
C                     END
C                     ELSE
C*
C* If error within *PSSR, then indicate *PSSR error
C*
C           '*PSSR   'DSPLY
C                     END
C*
C* No matter how the program got to the *PPSR, end the program
C*
C                     MOVE '1'       *INLR
C                     RETRN
C                     ENDSR
```

Related reference:

"Example in ILE C: Logging software errors (program API without pointers)" on page 13
This ILE C program calls the Log Software Error (QPDLOGER) API to perform first-failure data capture (FFDC) without using pointers.

Example in ILE RPG: Logging software error (program API without pointers):

This ILE RPG program performs a divide-by-zero operation to cause an exception. The exception is caught with RPG *PSSR support, which calls the Log Software Error (QPDLOGER) API to perform FFDC without using pointers.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*********************************************************************
F*
F* Program:     Demonstrate use of OPM based Log Software Error
F*
F* Language:    ILE RPG
F*
F* Description: This program performs a divide by 0 operation to
F*              cause an exception.  This exception is caught using
F*              RPG's *PSSR support, and the exception is then
F*              logged as a software error.
F*
F* APIs used:   QPDLOGER
F*
F*********************************************************************
D*
D* Include Error Code Parameter
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D* Misc. data elements
D*
Dfactor1          S              5B 0 INZ(10)
Dfactor2          S              5B 0 INZ(0)
Dresult           S              5B 0
Dline_nbr         S              9B 0
Ddata             DS          4096
Ddatapt           DS
D data_off                       9B 0
D data_len                       9B 0
```

```
Ddata#            S              9B 0
Dobjl             DS          2590
Dobjl#            S              9B 0
D*
D* Program status data structure
D*
DPSDS             SDS
D pgm_name                1     10
D status                 11     15  0
D src_line               21     28
D exception              40     46
D lib_name               81     90
C*
C* Attempt to divide by 0
C*
C     factor1       div       factor2       result
C*
C* Should not get here due to divide by 0 exception
C*
C                   move      '1'           *INLR
C                   return
C*
C* Program exception subroutine:
C*
C     *PSSR         BEGSR
C*
C* Make sure we are not catching an exception due to the *PSSR
C* subroutine itself
C*
C     switch        ifeq      ' '
C                   move      '1'           switch            1
C*
C* Set API error code to work in non-exception mode
C*
C                   eval      qusbprv = %size(qusec)
C*
C* Record line number where error happened
C*
C                   move      src_line      line_nbr
C*
C* Record the status code as data
C*
C                   movel     status        data
C*
C* Record where status located in data
C*
C                   eval      data_off = 1
C                   eval      data_len = 5
C                   eval      data# = 1
C*
C* For illustration purposes, dump the program object
C*
C                   eval      %SUBST(objl:1:30) = pgm_name             (1)
C                   eval      %SUBST(objl:31:30) = lib_name
C                   eval      %SUBST(objl:61:10) = '*PGM'
C                   eval      objl# = 1
C*
C* Call the Report Software Error API
C*
C                   call      'QPDLOGER'
C                   parm                    pgm_name
C                   parm      exception     msgid            12
C                   parm                    msgkey            4
C                   parm                    line_nbr
C                   parm      'Y'           joblog            1
C                   parm                    data
C                   parm                    datapt
```

```
C                   parm                    data#
C                   parm                    objl
C                   parm                    objl#
C                   parm                    qusec
C*
C* If an error on the API call, then indicate a terminal error
C*
C     qusbavl       ifgt      0
C     'Terminal err'dsply
C                   end
C                   else
C*
C* If error within *PSSR, then indicate *PSSR error
C*
C     '*PSSR error' dsply
C                   end
C*
C* No matter how the program got to the *PSSR, end the program
C*
C                   move      '1'           *inlr
C                   return
C                   endsr
```

Related reference:

"Example in ILE C: Logging software errors (program API without pointers)" on page 13
This ILE C program calls the Log Software Error (QPDLOGER) API to perform first-failure data capture (FFDC) without using pointers.

Example in ILE C: Reporting software errors (bindable API with pointers):

This ILE C program calls the Report Software Error (QpdReportSoftwareError) API to perform FFDC using pointers.

The ILE C program sets a pointer, as shown at (2), to point to the same location as in "Example in ILE C: Logging software errors (program API without pointers)" on page 13 at **(1)**.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*                                                                   */
/*Program Name: FFDCPGM2                                             */
/*                                                                   */
/*Program Language:  ILE C                                           */
/*                                                                   */
/*Description:  This program illustrates how to use APIs to log      */
/*              software errors using FFDC.                          */
/*                                                                   */
/*                                                                   */
/*Header Files Included:  except                                     */
/*                        stdio                                      */
/*                        string                                     */
/*                        qmhchgem                                   */
/*                        qpdsrvpg                                   */
/*                        qusec                                      */
/*                                                                   */
/*APIs Used:     QpdReportSoftwareError                              */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*                        System Includes                            */
/*********************************************************************/
#include <except.h>               /* from QSYSINC/H                  */
#include <stdio.h>                /* from QSYSINC/H                  */
```

```
#include <string.h>               /* from QSYSINC/H                */

/*****************************************************************/
/*                    Miscellaneous Includes                     */
/*****************************************************************/
#include <qmhchgem.h>
#include <qpdsrvpg.h>
#include <qusec.h>

/*****************************************************************/
/* Definitions used for developing key information for FFDC.     */
/*****************************************************************/
#define CHARACTER 'C'
#define MAX_KEYS 3
#define MESSAGE "MSG"
#define MESSAGE_LEN 7
#define MSG_SYMPTOM_LEN 3

/*****************************************************************/
/*                          Structures                           */
/*****************************************************************/
typedef struct {
    void  *parm1;
    void  *parm2;
    char  *pgm_name;
    int    pgm_name_size;
} ffdc_info_t;

/*****************************************************************/
/*                          Prototypes                           */
/*****************************************************************/
void UNEXPECTED_HDLR(_INTRPT_Hndlr_Parms_T *);

/*****************************************************************/
/*   FUNCTION NAME:  main                                        */
/*                                                               */
/*   FUNCTION:       Generates exception and then passes control */
/*                   to exception handler.                       */
/*                                                               */
/*   INPUT:          Two character strings.                      */
/*                                                               */
/*   OUTPUT:         NONE                                        */
/*                                                               */
/*   EXCEPTIONS:     CPFxxxx - All unexpected CPF exceptions     */
/*                   MCHxxxx - All unexpected MCH exceptions     */
/*                                                               */
/*****************************************************************/
void main(int argc, char *argv[])
{
  /***************************************************************/
  /* NOTE:  argv will contain the parameters passed in to this   */
  /*        function.  In this case, two parameters are passed   */
  /*        in.                                                  */
  /***************************************************************/
  /***************************************************************/
  /* The argv parameter contains the parameters that were passed as  */
  /* character arrays.  argv[0] contains the program name, and the   */
  /* parameter(s) starts with argv[1].                           */
  /***************************************************************/

  char *nulptr;                   /* Pointer used to generate error  */
  char pgm_name[30];              /* Program name                    */
  volatile ffdc_info_t ffdc_info; /* FFDC info for unexpected error  */

  /***************************************************************/
  /* Set up FFDC information for unexpected error.               */
  /***************************************************************/
```

```
  ffdc_info.parm1 = argv[1];
  ffdc_info.parm2 = argv[2];
  ffdc_info.pgm_name = pgm_name;
  memcpy(pgm_name, argv[0], strlen(argv[0]));
  ffdc_info.pgm_name_size = strlen(argv[0]);

  /*******************************************************************/
  /* Enable the exception handler, and pass ffdc_info into the       */
  /* exception handler via the communications area so that data      */
  /* can be used for FFDC.                                           */
  /*******************************************************************/

#pragma exception_handler (UNEXPECTED_HDLR, ffdc_info, 0, _C2_MH_ESCAPE)

  /*******************************************************************/
  /* Set the pointer to null, then try to increment.  This will      */
  /* generate an MCH3601 error that will be trapped by the           */
  /* unexpected handler.                                             */
  /*******************************************************************/
  nulptr = NULL;
  nulptr++;

#pragma disable_handler

} /* main */

/*********************************************************************/
/*   FUNCTION NAME:  UNEXPECTED_HDLR                                 */
/*                                                                   */
/*   FUNCTION:       Handle unexpected exception.  This exception    */
/*                   handler is used to log the software error via   */
/*                   FFDC.                                           */
/*                                                                   */
/*   INPUT:          Interrupt handler information                   */
/*                                                                   */
/*   OUTPUT:         NONE                                            */
/*                                                                   */
/*   EXCEPTIONS:     CPFxxxx - All unexpected CPF exceptions         */
/*                   MCHxxxx - All unexpected MCH exceptions         */
/*                                                                   */
/*********************************************************************/
void UNEXPECTED_HDLR(_INTRPT_Hndlr_Parms_T *errmsg)
{

  int                    i = 0,
                         MsgLen = 0,
                         number_of_keys = 0;
  char                   pgm_name[30],
                         context_name[30],
                         lib_name[5],
                         symptom_msg_data[MESSAGE_LEN],
                         symptom_msg_keyword[MSG_SYMPTOM_LEN];
  ffdc_info_t            *ffdc_info;
  Qpd_Data_t             data_key,
                         data_key2;
  Qpd_Key_Pointer_t      ffdc_keys[MAX_KEYS];
  Qpd_Suspected_Module_t module_key;
  Qpd_Symptom_t          symptom_msg_key;
  Qus_EC_t               ErrorCode;

  ErrorCode.Bytes_Provided = 0;

  /*******************************************************************/
  /* Getting pointer in local storage to the Communications Area.    */
  /*******************************************************************/
  ffdc_info = (ffdc_info_t *)(errmsg->Com_Area);
```

```
  /*****************************************************************/
  /* Need to notify message handler that we will handle the error.   */
  /* Leave the message in the job log, just mark it handled.         */
  /*****************************************************************/
  QMHCHGEM(&(errmsg->Target),            /* Invocation pointer      */
           0,                            /* Call stack counter      */
           (char *)&errmsg->Msg_Ref_Key,/* Message key             */
           "*HANDLE   ",                 /* Modification option     */
           "",                           /* Reply text              */
           0,                            /* Reply text length       */
           &ErrorCode);                  /* Error code              */

  /*****************************************************************/
  /* Initialize module suspected key for FFDC.                       */
  /*****************************************************************/
  ffdc_keys[number_of_keys++].Suspected_Module = &module_key;
  module_key.Key = Qpd_Suspected_Module;
  module_key.Module_Name_Length = ffdc_info->pgm_name_size;
  module_key.Library_Name_Length = 7;
  module_key.Module_Name = pgm_name;
  memcpy(pgm_name, ffdc_info->pgm_name, ffdc_info->pgm_name_size);
  module_key.Library_Name = lib_name;
  memcpy(lib_name, "TESTLIB", 7);

  /*****************************************************************/
  /* Initialize symptom keys for FFDC.                               */
  /*****************************************************************/
  ffdc_keys[number_of_keys++].Symptom = &symptom_msg_key;
  symptom_msg_key.Key = Qpd_Symptom;
  symptom_msg_key.Keyword_Length = MSG_SYMPTOM_LEN;
  symptom_msg_key.Data_Length = MESSAGE_LEN;
  symptom_msg_key.Data_Type = CHARACTER;
  memcpy(symptom_msg_keyword, MESSAGE, MSG_SYMPTOM_LEN);
  symptom_msg_key.Keyword = symptom_msg_keyword;
  memcpy(symptom_msg_data, errmsg->Msg_Id, MESSAGE_LEN);
  symptom_msg_key.Data = symptom_msg_data;

  /*****************************************************************/
  /* Parameter 1 information                                         */
  /*****************************************************************/
  ffdc_keys[number_of_keys++].Data = &data_key;
  data_key.Key = Qpd_Data;
  data_key.Data_Length = sizeof(char *);
  data_key.Data_Id = 1;
  data_key.Data = ffdc_info->parm1;                              (2)


  /*****************************************************************/
  /* Parameter 2 information                                         */
  /*****************************************************************/
  ffdc_keys[number_of_keys++].Data = &data_key2;
  data_key2.Key = Qpd_Data;
  data_key2.Data_Length = sizeof(char *);
  data_key2.Data_Id = 1;
  data_key2.Data = ffdc_info->parm2;

  /*****************************************************************/
  /* Call QpdReportSoftwareError to perform FFDC.                    */
  /*****************************************************************/
  ErrorCode.Bytes_Provided = sizeof(ErrorCode);
  QpdReportSoftwareError(ffdc_keys,
                         &number_of_keys,
                         &ErrorCode);

} /* UNEXPECTED_HDLR */
```

Example in ILE COBOL: Reporting software errors (bindable API with pointers):

One ILE COBOL program registers an error handler. After the successful completion of the registration, this program creates a data decimal error. The error causes the error handler, the other ILE COBOL program, to call the Report Software Error (QpdReportSoftwareError) API using pointers.

In this example, the "CBLERR2 program" registers the error handler and causes the error. The "ERRHDL2 program" on page 31 sets a pointer, as shown at (2), to point to the same location as in "Example in ILE C: Logging software errors (program API without pointers)" on page 13 at **(1)** and calls the QpdReportSoftwareError API to perform FFDC.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

CBLERR2 program

```
 PROCESS NOMONOPRC.
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*  Program:      Register an ILE COBOL Error Handler
*                Cause a decimal data exception to demonstrate
*                   logging of software errors
*
*  Language:     ILE COBOL
*
*  Description:  This program registers an ILE COBOL Error
*                Handler.  After the successful completion of
*                the registration of the error handler, this     ,
*                program creates a decimal data error. This
*                exception causes the error handler to be
*                called which then logs the software error.
*
*  APIs Used:    QlnSetCobolErrorHandler
*
*
*****************************************************************
*
*****************************************************************
 PROGRAM-ID. CBLERR2.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QCBLLESRC.
*
* Miscellaneous elements
*
 01  MISC.
     05  Y               PIC S9(09) VALUE 0.
 01  ERROR-HANDLER       PROCEDURE-POINTER.
 01  OLD-ERROR-HANDLER   PROCEDURE-POINTER.
 01  NUMERIC-GROUP.
     05  X               PIC  9(03).
*
```

```
* Beginning of mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Register the COBOL Error Handler.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
* API for the parameter.
*
     MOVE 16 TO BYTES-PROVIDED.
*
* Set ERROR-HANDLER procedure pointer to entry point of
* ERRHDL1 *PGM
*
     SET ERROR-HANDLER TO ENTRY LINKAGE PROGRAM "ERRHDL2".
*
*
* Call the API to register the exit point.
*
     CALL "QlnSetCobolErrorHandler" USING ERROR-HANDLER,
                                    OLD-ERROR-HANDLER,
                                    QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE > 0
                            DISPLAY "Error setting handler",
                            STOP RUN.
*
* If the call to register an error handler is successful, then
* cause a the data decimal error (X is initialized to blanks).
*
     ADD X TO Y.
*
* Should not get here due to data decimal error
*
     STOP RUN.
*
* End of MAINLINE
*
```

ERRHDL2 program

```
 PROCESS NOMONOPRC.
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*  Program:      Log a software error
*
*  Language:     ILE COBOL
*
*  Description:  This program receives control for exceptions
*                within a COBOL run unit.  This program is used
*                in conjunction with CBLERR2.                    ,
*                Any exception causes this error handler to be
*                called which then logs the software error.
*
*  APIs Used:    QpdReportSoftwareError
```

```
*
****************************************************************
*
****************************************************************
 PROGRAM-ID. ERRHDL2.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QCBLLESRC.
*
* QpdReportSoftwareError include
*
 COPY QPDSRVPG OF QSYSINC-QCBLLESRC.
*
* Miscellaneous elements
*
 01  MISC.
     05  NBR-OF-RECORDS   PIC S9(09) BINARY VALUE 0.
     05  MSG-KEYWORD      PIC  X(03) VALUE "MSG".
 01  PROBLEM-RECORDS.
     05  PROBLEM-POINTER  POINTER OCCURS 100 TIMES.
 LINKAGE SECTION.
 01  CBL-EXCEPTION-ID     PIC  X(07).
 01  VALID-RESPONSES      PIC  X(06).
 01  PGM-IN-ERROR.
     05  PGM-NAME         PIC  X(10).
     05  LIB-NAME         PIC  X(10).
 01  SYS-EXCEPTION-ID     PIC  X(07).
 01  MESSAGE-TEXT         PIC  X(01).
 01  MESSAGE-LENGTH       PIC S9(09) BINARY.
 01  SYS-OPTION           PIC  X(01).
 01  ERR-MODULE-NAME      PIC  X(10).
 01  CBL-PGM-NAME         PIC X(256).
*
* Beginning of mainline
*
 PROCEDURE DIVISION USING CBL-EXCEPTION-ID,
                          VALID-RESPONSES,
                          PGM-IN-ERROR,
                          SYS-EXCEPTION-ID,
                          MESSAGE-LENGTH,
                          SYS-OPTION,
                          MESSAGE-TEXT,
                          ERR-MODULE-NAME,
                          CBL-PGM-NAME.
 MAIN-LINE.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
* API for the parameter.
*
     MOVE 16 TO BYTES-PROVIDED.
*
* Record the COBOL Program and Library names
```

```
*
     MOVE 101 TO KEY-FIELD OF QPD-SUSPECTED-PROGRAM.
     MOVE 10 TO PROGRAM-NAME-LENGTH OF QPD-SUSPECTED-PROGRAM.
     MOVE 10 TO LIBRARY-NAME-LENGTH OF QPD-SUSPECTED-PROGRAM.
     SET PROGRAM-NAME OF QPD-SUSPECTED-PROGRAM                          (2)
                      TO ADDRESS OF PGM-NAME OF PGM-IN-ERROR.
     SET LIBRARY-NAME OF QPD-SUSPECTED-PROGRAM
                      TO ADDRESS OF LIB-NAME OF PGM-IN-ERROR.
     ADD 1 TO NBR-OF-RECORDS.
     SET PROBLEM-POINTER (NBR-OF-RECORDS) TO
                         ADDRESS OF QPD-SUSPECTED-PROGRAM.
*
* Record the message id
*
     MOVE 200 TO KEY-FIELD OF QPD-SYMPTOM.
     MOVE 3 TO KEYWORD-LENGTH OF QPD-SYMPTOM.
     MOVE 7 TO DATA-LENGTH OF QPD-SYMPTOM.
     MOVE "C" TO DATA-TYPE OF QPD-SYMPTOM.
     SET KEYWORD OF QPD-SYMPTOM TO ADDRESS OF MSG-KEYWORD.
     SET DATA-FIELD OF QPD-SYMPTOM TO ADDRESS OF SYS-EXCEPTION-ID.
     ADD 1 TO NBR-OF-RECORDS.
     SET PROBLEM-POINTER (NBR-OF-RECORDS) TO
                         ADDRESS OF QPD-SYMPTOM.
*
* For illustration purposes, dump the program object
*
     MOVE 302 TO KEY-FIELD OF QPD-NAMED-SYSTEM-OBJECT.
     MOVE PGM-NAME OF PGM-IN-ERROR
                   TO OBJECT-NAME OF QPD-NAMED-SYSTEM-OBJECT.
     MOVE LIB-NAME OF PGM-IN-ERROR
                   TO OBJECT-LIBRARY OF QPD-NAMED-SYSTEM-OBJECT.
     MOVE "*PGM" TO OBJECT-TYPE OF QPD-NAMED-SYSTEM-OBJECT.
     ADD 1 TO NBR-OF-RECORDS.
     SET PROBLEM-POINTER (NBR-OF-RECORDS) TO
                         ADDRESS OF QPD-NAMED-SYSTEM-OBJECT.
*
* Call the API to log the software error.
*
     CALL "QpdReportSoftwareError" USING PROBLEM-RECORDS,
                                         NBR-OF-RECORDS,
                                         QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE > 0 DISPLAY "Cannot log error".
*
* End the current run unit
*
     MOVE "C" TO SYS-OPTION.
     STOP RUN.
*
* End of MAINLINE
*
```

Example in ILE RPG: Reporting software errors (bindable API with pointers):

This ILE RPG program performs a divide-by-zero operation to cause an exception. The exception is caught with RPG *PSSR support, which calls the Report Software Error (QpdReportSoftwareError) API using pointers.

The ILE RPG program sets a pointer, as shown at (2), to point to the same location as in "Example in ILE C: Logging software errors (program API without pointers)" on page 13 at **(1)** and calls the QpdReportSoftwareError API to perform FFDC.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*****************************************************************
F*
F* Program:     Demonstrate use of ILE-based Report Software Error
F*
F* Language:    ILE RPG
F*
F* Description: This program performs a divide-by-0 operation to
F*              cause an exception.  This exception is caught using
F*              RPGs *PSSR support, and the exception is then logged
F*              as a software error.
F*
F* APIs used:   QpdReportSoftwareError
F*
F*****************************************************************
D*
D* Include Error Code Parameter
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D* Include API structures and constants
D*
D/COPY QSYSINC/QRPGLESRC,QPDSRVPG
D*
D* Array of problem record description pointers and index to array
D*
Dpdr              S               *   dim(20)
Dx                S              5B 0 INZ(1)
D*
D* Misc. data elements
D*
Dfactor1          S              5B 0 INZ(10)
Dfactor2          S              5B 0 INZ(0)
Dresult           S              5B 0
Drc               S              2    INZ('RC')
D*
D* Program status data structure
D*
DPSDS            SDS
D pgm_name                1     10
D status                 11     15  0
D src_line               21     28
D exception              40     46
D lib_name               81     90
C*
C* Attempt to divide by 0
C*
C     factor1       div       factor2       result
C*
C* Should not get here due to divide-by-0 exception
C*
C                   move      '1'           *INLR
C                   return
C*
C* Program exception subroutine:
C*
C     *PSSR         BEGSR
C*
C* Make sure we are not catching an exception due to the *PSSR
C* subroutine itself
C*
```

```
C     switch        ifeq      ' '
C                   move      '1'           switch            1
C*
C* Set API error code to work in nonexception mode
C*
C                   eval      qusbprv = %size(qusec)
C*
C* Record the suspected program and library name
C*
C                   eval      qpdk01 = 101
C                   eval      qpdpgmnl = %SIZE(pgm_name)
C                   eval      qpdlibnl = %SIZE(lib_name)
C                   eval      qpdpgmn = %ADDR(pgm_name)        (2)
C                   eval      qpdlibn = %ADDR(lib_name)
C*
C*       and record the key:
C*
C                   eval      pdr(x) = %addr(qpdspgm)
C                   eval      x = x + 1
C*
C* Record the failing source statement number
C*
C                   eval      qpdk07 = 200
C                   eval      qpdkl = %SIZE(rc)
C                   eval      qpddl = %SIZE(src_line)
C                   eval      qpddt = 'C'
C                   eval      qpdk08 = %ADDR(rc)
C                   eval      qpdd = %ADDR(src_line)
C*
C*       and record the key:
C*
C                   eval      pdr(x) = %addr(qpds)
C                   eval      x = x + 1
C*
C* Record the status code as data
C*
C                   eval      qpdk11 = 301
C                   eval      qpddl00 = %SIZE(status)
C                   eval      qpddi = 1
C                   eval      qpdd00 = %ADDR(status)
C*
C*       and record the key:
C*
C                   eval      pdr(x) = %addr(qpds)
C                   eval      x = x + 1
C*
C* For illustration purposes, dump the program object
C*
C                   eval      qpdk12 = 302
C                   eval      qpdobjn = pgm_name
C                   eval      qpdobjlib = lib_name
C                   eval      qpdobjt = '*PGM'
C*
C*       and record the key:
C*
C                   eval      pdr(x) = %addr(qpdnsot)
C                   eval      x = x + 1
C*
C* Call the Report Software Error API
C*
C                   callb     qpdrse
C                   parm                    pdr
C                   parm                    x
C                   parm                    qusec
C*
C* If an error on the API call, then indicate a terminal error
C*
```

```
C     qusbavl       ifgt      0
C     'Terminal err'dsply
C                   end
C                   else
C*
C* If error within *PSSR, then indicate *PSSR error
C*
C     '*PSSR error' dsply
C                   end
C*
C* No matter how the program got to the *PSSR, end the program
C*
C                   move      '1'           *inlr
C                   return
C                   endsr
```

APIs for the UNIX-type environment

UNIX-type APIs, which include the socket APIs and the integrated file system APIs, support an open environment on the system.

The socket functions and the integrated file system reduce the amount of effort to move UNIX applications to the system.

The integrated file system is a function of the IBM i operating system that supports stream input/output and storage management similar to personal computers and UNIX operating systems. It also provides an integrating structure over all information stored on the system.

The naming conventions for the UNIX-type APIs are determined by industry standards organizations.

Related reference:

UNIX-Type APIs

Examples: UNIX-type APIs:

These example programs use several integrated file system functions.

Each program performs the following operations:

(1) Uses the **getuid()** function to determine the real user ID (uid).

(2) Uses the **getcwd()** function to determine the current directory.

(3) Uses the **open()** function to create a file. The owner (the person who created the file) is given read, write, and execute authority to the file.

(4) Uses the **write()** function to write a byte string to the file. The file is identified by the file descriptor that was provided in the open operation (**(3)**).

(5) Uses the **close()** function to close the file.

(6) Uses the **open()** function to open the file for read only.

(7) Uses the **read()** function to read a byte string from the file. The file is identified by the file descriptor that was provided in the open operation (**(6)**).

(8) Uses the **close()** function to close the file.

(9) Uses the **unlink()** function to remove the link to the file.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 571.

Example in ILE C: Using the integrated file system

```
/*********************************************************************/
/*                                                                   */
/* Language:            ILE C                                        */
/*                                                                   */
/* Description:         Demonstrate use of integrated file system    */
/*                      from ILE C                                   */
/*                                                                   */
/*********************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>


#define BUFFER_SIZE       2048
#define TEST_FILE         "test.file"
#define TEST_DATA         "Hello World!"
#define USER_ID           "user_id_"


char  InitialFile[BUFFER_SIZE];
char  InitialDirectory[BUFFER_SIZE] = ".";
char  Buffer[32];
int   FilDes = -1;
int   BytesRead;
int   BytesWritten;
uid_t UserID;


void CleanUpOnError(int level)
{
   printf("Error encountered, cleaning up.\n");
   switch ( level )
      {
       case 1:
           printf("Could not get current working directory.\n");
           break;
       case 2:
           printf("Could not create file %s.\n",TEST_FILE);
           break;
       case 3:
           printf("Could not write to file %s.\n",TEST_FILE);
           close(FilDes);
           unlink(TEST_FILE);
           break;
       case 4:
           printf("Could not close file %s.\n",TEST_FILE);
           close(FilDes);
           unlink(TEST_FILE);
           break;
       case 5:
           printf("Could not open file %s.\n",TEST_FILE);
           unlink(TEST_FILE);
           break;
       case 6:
           printf("Could not read file %s.\n",TEST_FILE);
           close(FilDes);
           unlink(TEST_FILE);
           break;
       case 7:
           printf("Could not close file %s.\n",TEST_FILE);
           close(FilDes);
           unlink(TEST_FILE);
           break;
```

```
       case 8:
           printf("Could not unlink file %s.\n",TEST_FILE);
           unlink(TEST_FILE);
           break;
       default:
           break;
      }
   printf("Program ended with Error.\n"\
          "All test files and directories may not have been removed.\n");
}


int main ()
{
(1)
/* Get and print the real user id with the getuid() function. */
   UserID = getuid();
   printf("The real user id is %u. \n",UserID);

(2)
/* Get the current working directory and store it in InitialDirectory. */
   if ( NULL == getcwd(InitialDirectory,BUFFER_SIZE) )
      {
      perror("getcwd Error");
      CleanUpOnError(1);
      return 0;
      }
   printf("The current working directory is %s. \n",InitialDirectory);

(3)
/* Create the file TEST_FILE for writing, if it does not exist.
   Give the owner authority to read, write, and execute. */
   FilDes = open(TEST_FILE, O_WRONLY | O_CREAT | O_EXCL, S_IRWXU);
   if ( -1 == FilDes )
      {
      perror("open Error");
      CleanUpOnError(2);
      return 0;
      }
   printf("Created %s in directory %s.\n",TEST_FILE,InitialDirectory);

(4)
/* Write TEST_DATA to TEST_FILE via FilDes */
   BytesWritten = write(FilDes,TEST_DATA,strlen(TEST_DATA));
   if ( -1 == BytesWritten )
      {
      perror("write Error");
      CleanUpOnError(3);
      return 0;
      }
   printf("Wrote %s to file %s.\n",TEST_DATA,TEST_FILE);

(5)
/* Close TEST_FILE via FilDes */
   if ( -1 == close(FilDes) )
      {
      perror("close Error");
      CleanUpOnError(4);
      return 0;
      }
   FilDes = -1;
   printf("File %s closed.\n",TEST_FILE);


(6)
/* Open the TEST_FILE file for reading only. */
   if ( -1 == (FilDes = open(TEST_FILE,O_RDONLY)) )
```

```
      {
      perror("open Error");
      CleanUpOnError(5);
      return 0;
      }
   printf("Opened %s for reading.\n",TEST_FILE);


(7)
/* Read from the TEST_FILE file, via FilDes, into Buffer. */
   BytesRead = read(FilDes,Buffer,sizeof(Buffer));
   if ( -1 == BytesRead )
      {
      perror("read Error");
      CleanUpOnError(6);
      return 0;
      }
   printf("Read %s from %s.\n",Buffer,TEST_FILE);
   if ( BytesRead != BytesWritten )
      {
      printf("WARNING: the number of bytes read is "\
             "not equal to the number of bytes written.\n");
      }

(8)
/* Close the TEST_FILE file via FilDes. */
   if ( -1 == close(FilDes) )
      {
      perror("close Error");
      CleanUpOnError(7);
      return 0;
      }
   FilDes = -1;
   printf("Closed %s.\n",TEST_FILE);

(9)
/* Unlink the file TEST_FILE */
   if ( -1 == unlink(TEST_FILE) )
      {
      perror("unlink Error");
      CleanUpOnError(8);
      return 0;
      }
   printf("Unlinking file %s.\n",TEST_FILE);


   printf("Program completed successfully.\n");
   return 0;
}
```

Example in ILE COBOL: Using the integrated file system

```
       PROCESS NOMONOPRC.
       IDENTIFICATION DIVISION.
      *****************************************************************
      *****************************************************************
      *
      *  Language:     COBOL
      *
      *  Description:  Demonstrate use of integrated file system
      *                from ILE COBOL
      *
      *****************************************************************
      *
      *****************************************************************
       PROGRAM-ID. IFS.
       INPUT-OUTPUT SECTION.
```

```
 FILE-CONTROL.
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
 DATA DIVISION.
*
 FILE SECTION.
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
*
 WORKING-STORAGE SECTION.
*
* Report lines
*
 01  REALID.
     05  PRT-TEXT        PIC  X(20) VALUE "The real user id is ".
     05  USER            PIC  X(12).
 01  CURDIR.
     05  PRT-TEXT       PIC  X(21) VALUE "Current directory is ".
     05  INITIALDIR      PIC  X(100).
 01  NEWFIL.
     05  PRT-TEXT        PIC  X(20) VALUE "Created file:       ".
     05  FILENAME        PIC  X(100).
 01  DATAIN.
     05  PRT-TEXT        PIC  X(20) VALUE "Successfully read:  ".
     05  DATA-READ       PIC  X(100).
 01  ERRLIN.
     05  PRT-TEXT        PIC  X(20) VALUE "The errno value is: ".
     05  ERRVAL          PIC  X(12).
*
* Miscellaneous elements
*
 01  BUFFER              PIC  X(32767).
 01  LENGTH-OF-BUFFER    PIC S9(09) BINARY VALUE 32767.
 01  TESTFILE.
     05 TEST-FILE        PIC  X(09) VALUE "test.file".
     05 NULL-TERMINATE   PIC  X(01) VALUE LOW-VALUE.
 01  OFLAG               PIC  X(04) VALUE X"0000001A".
 01  OFLAG-READ          PIC  X(04) VALUE X"00000001".
 01  OMODE               PIC  X(04) VALUE X"000001C0".
 01  TEST-DATA           PIC  X(12) VALUE "Hello World!".
 01  SIZE-TEST-DATA      PIC S9(09) BINARY VALUE 12.
 01  FILE-DESCRIPTOR     PIC S9(09) BINARY.
 01  BYTES-READ          PIC S9(09) BINARY.
 01  BYTES-WRITTEN       PIC S9(09) BINARY.
 01  RETURN-INT          PIC S9(09) BINARY.
 01  RETURN-PTR          POINTER.
*
* Beginning of mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
     OPEN OUTPUT LISTING.
*
* Get and print the real user id with the getuid function.
*
     CALL "getuid" GIVING RETURN-INT.
*
* Check for error and report status.
*
     IF RETURN-INT = -1  MOVE "Error getting real user id"
                               TO LIST-LINE,
                         PERFORM ERROR-FOUND,
            ELSE         MOVE RETURN-INT TO USER,
                         WRITE LIST-LINE FROM REALID.
*
```

```
* Get the current working directory and store it in BUFFER
*
     CALL "getcwd" USING BY VALUE ADDRESS OF BUFFER,
                         BY VALUE LENGTH-OF-BUFFER,
                         GIVING RETURN-PTR.
*
* Check for error and report status.
*
     IF RETURN-PTR = NULL MOVE "Error getting real current dir"
                               TO LIST-LINE,
                         PERFORM ERROR-FOUND,
            ELSE         MOVE BUFFER TO INITIALDIR,
                         WRITE LIST-LINE FROM CURDIR.
*
* Create the file test.file for writing.  If it does not exist,
* give the owner authority to read, write, and execute.
*
     CALL "open"   USING BY VALUE ADDRESS OF TESTFILE,
                         BY VALUE OFLAG,
                         BY VALUE OMODE,
                         GIVING FILE-DESCRIPTOR.
*
* Check for error and report status.
*
     IF FILE-DESCRIPTOR = -1  MOVE "Could not create file"
                               TO LIST-LINE,
                         PERFORM ERROR-FOUND,
            ELSE         MOVE TEST-FILE TO FILENAME,
                         WRITE LIST-LINE FROM NEWFIL.
*
* Write TEST-DATA to test.file via file descriptor from open
*
     CALL "write"  USING BY VALUE FILE-DESCRIPTOR,
                         BY VALUE ADDRESS OF TEST-DATA,
                         BY VALUE SIZE-TEST-DATA,
                         GIVING BYTES-WRITTEN.
*
* Check for error and report status.
*
     IF BYTES-WRITTEN = -1  MOVE "Could not write to file"
                               TO LIST-LINE,
                         PERFORM ERROR-FOUND,
            ELSE         MOVE "Wrote to file successfully"
                               TO LIST-LINE,
                         WRITE LIST-LINE.
*
* Close test.file via file descriptor
*
     CALL "close"  USING BY VALUE FILE-DESCRIPTOR,
                         GIVING RETURN-INT.
*
* Check for error and report status.
*
     IF RETURN-INT    = -1  MOVE "Could not close file"
                               TO LIST-LINE,
                         PERFORM ERROR-FOUND,
            ELSE         MOVE "Successfully closed file"
                               TO LIST-LINE,
                         WRITE LIST-LINE.
*
* Open the file test.file for reading.
*
     CALL "open"   USING BY VALUE ADDRESS OF TESTFILE,
                         BY VALUE OFLAG-READ,
                         GIVING FILE-DESCRIPTOR.
*
* Check for error and report status.
```

```
*
     IF FILE-DESCRIPTOR = -1  MOVE "Could not open file"
                              TO LIST-LINE,
                         PERFORM ERROR-FOUND,
            ELSE         MOVE "File open successful"
                              TO LIST-LINE,
                         WRITE LIST-LINE.
*
* Read from test.file via file descriptor from open
*
     CALL "read"   USING BY VALUE FILE-DESCRIPTOR,
                         BY VALUE ADDRESS OF BUFFER,
                         BY VALUE LENGTH-OF-BUFFER,
                         GIVING BYTES-READ.
*
* Check for error and report status.
*
     IF BYTES-READ = -1  MOVE "Read failed"
                              TO LIST-LINE,
                         PERFORM ERROR-FOUND,
            ELSE  IF BYTES-READ = BYTES-WRITTEN
                         MOVE BUFFER TO DATA-READ,
                         WRITE LIST-LINE FROM DATAIN,
                  ELSE MOVE "Data Truncation on Read"
                            TO LIST-LINE,
                         PERFORM ERROR-FOUND.
*
* Close test.file via file descriptor
*
     CALL "close"  USING BY VALUE FILE-DESCRIPTOR,
                         GIVING RETURN-INT.
*
* Check for error and report status.
*
     IF RETURN-INT    = -1  MOVE "Could not close file"
                              TO LIST-LINE,
                         PERFORM ERROR-FOUND,
            ELSE         MOVE "Successfully closed file"
                              TO LIST-LINE,
                         WRITE LIST-LINE.
*
* Unlink test.file
*
     CALL "unlink"  USING BY VALUE ADDRESS OF TESTFILE,
                         GIVING RETURN-INT.
*
* Check for error and report status.
*
     IF RETURN-INT    = -1  MOVE "Unlink of file failed"
                              TO LIST-LINE,
                         PERFORM ERROR-FOUND,
            ELSE         MOVE "Unlink of file successful"
                              TO LIST-LINE,
                         WRITE LIST-LINE.
*
     MOVE "Program run is successful" TO LIST-LINE.
     WRITE LIST-LINE.
     STOP RUN.
*
* End of MAINLINE
*
*
* Common error reporting subroutine
*
* If errors occur, the Integrated File System exports the
* variable 'errno' to assist in determining the problem.  As
* 'errno' is lowercase, ILE COBOL cannot directly import this
```

```
 * variable and must use a C module to access it.  If the
 * developer has ILE C available, the following sample C code
 * will import 'errno' and make it available to the COBOL
 * application
 *
 *       #include <errno.h>
 *       int geterrno()
 *       {
 *         return errno;
 *       }
 *
 * To activate this C module remove the comment identifiers
 * following the WRITE statement and remove the comment
 * identifier from the geterrno declaration in the Configuration
 * Section.  Definitions for the returned errno are found in
 * file QSYSINC/SYS member ERRNO.
 *
  ERROR-FOUND.
      WRITE LIST-LINE.
 *    CALL "geterrno" GIVING RETURN-INT.
 *    MOVE RETURN-INT TO ERRVAL.
 *    WRITE LIST-LINE FROM ERRLIN.
      STOP RUN.
```

Example in ILE RPG: Using the integrated file system

```
F*****************************************************************
F*
F*   Language:     ILE RPG
F*
F*   Description:  Demonstrate use of integrated file system
F*                 from ILE RPG
F*
F*****************************************************************
FQSYSPRT   O    F  132        PRINTER
D*
D* Prototype the Integrated File System APIs
D*
Dgetuid           PR             9B 0 EXTPROC('getuid')
Dgetcwd           PR              *   EXTPROC('getcwd')
D                                 *   VALUE
D                                9B 0 VALUE
Dopen             PR             9B 0 EXTPROC('open')
D                                 *   VALUE
D                                4A   VALUE
D                                4A   VALUE
Dwrite            PR             9B 0 EXTPROC('write')
D                                9B 0 VALUE
D                                 *   VALUE
D                                9B 0 VALUE
Dclose            PR             9B 0 EXTPROC('close')
D                                9B 0 VALUE
Dopen2            PR             9B 0 EXTPROC('open')
D                                 *   VALUE
D                                4A   VALUE
Dread             PR             9B 0 EXTPROC('read')
D                                9B 0 VALUE
D                                 *   VALUE
D                                9B 0 VALUE
Dunlink           PR             9B 0 EXTPROC('unlink')
D                                 *   VALUE
D*
D* errno prototype; see error subroutine for further information
D*
D*errno           PR             9B 0 EXTPROC('geterrno')
DUser             S              12A
DBuffer           S           32767A
```

```
DReturnPtr        S               *
DReturnInt        S              9B 0
DFileDesc         S              9B 0
Dtest_file        S            2048A   INZ('test.file')
DInitialDir       S            2048A
Dtest_data        S              12A   INZ('Hello World!')
DBytesWrt         S              9B 0
DBytesRead        S              9B 0
DFileName         S            2049A
DPrintLine        S             100A
DNull             C                    CONST(X'00')
C*
C* Get and print the real user id with the getuid function.
C*
C                   eval      ReturnInt = getuid
C*
C* Check for error and report status.
C*
C                   if        ReturnInt = -1
C                   eval      PrintLine = 'Error getting real user id'
C                   exsr      error
C                   eval      *INLR = '1'
C                   return
C                   else
C                   move      ReturnInt     User
C                   eval      PrintLine = 'The real user id is '
C                                 + %TRIML(User)
C                   except
C                   endif
C*
C* Get the current working directory and store it in Buffer.
C*
C                   eval      ReturnPtr=getcwd(%ADDR(Buffer)
C                               : %SIZE(Buffer))
C*
C* Check for error and report status.
C*
C                   if        ReturnPtr = *NULL
C                   eval      PrintLine = 'Error getting current directory'
C                   exsr      error
C                   eval      *INLR = '1'
C                   return
C                   else
C*
C* Print current directory name remembering to scan for null terminator.
C*
C     Null          scan      Buffer        NullFound         5 0
C                   eval      InitialDir = %SUBST(Buffer&#58;1&#58;NullFound)
C                   eval      PrintLine = 'Current Directory is '
C                               + InitialDir
C                   except
C                   endif
C*
C* Create the file TEST_FILE for writing.  If it does not exist,
C* give the owner authority to read, write, and execute.
C*
C                   eval      FileName = %TRIMR(test_file) + Null
C                   eval      FileDesc = open(%ADDR(FileName)
C                               : x'0000001A' : x'000001C0')
C*
C* Check for error and report status.
C*
C                   if        FileDesc = -1
C                   eval      PrintLine = 'Could not create file'
C                   exsr      error
C                   eval      *INLR = '1'
C                   return
```

```
C                   else
C                   eval      PrintLine = 'File '
C                               + %TRIMR(test_file)
C                               + ' created successfully'
C                   except
C                   end
C*
C* Write test_data to test_file via FileDesc returned by open
C*
C                   eval      BytesWrt = write(FileDesc
C                               : %ADDR(Test_Data)
C                               : %SIZE(Test_Data))
C*
C* Check for error and report status.  If an error occurs,
C* attempt cleanup.
C*
C                   if        BytesWrt = -1
C                   eval      PrintLine = 'Could not write to file'
C                   exsr      error
C                   eval      ReturnInt = close(FileDesc)
C                   eval      ReturnInt = unlink(%ADDR(FileName))
C                   eval      *INLR = '1'
C                   return
C                   else
C                   eval      PrintLine = 'Wrote to '
C                               + %TRIMR(test_file)
C                               + ' successfully'
C                   except
C                   endif
C*
C* Close test_file via FileDesc
C*
C                   eval      ReturnInt = close(FileDesc)
C*
C* Check for error and report status.  If an error occurs,
C* attempt cleanup.
C*
C                   if        ReturnInt = -1
C                   eval      PrintLine = 'Could not close file'
C                   exsr      error
C                   eval      ReturnInt = close(FileDesc)
C                   eval      ReturnInt = unlink(%ADDR(FileName))
C                   eval      *INLR = '1'
C                   return
C                   else
C                   eval      PrintLine = 'File '
C                               + %TRIMR(test_file)
C                               + ' closed successfully'
C                   except
C                   endif
C*
C* Open the file for read only
C*
C                   eval      FileDesc = open2(%ADDR(FileName)
C                               : x'00000001')
C*
C* Check for error and report status.  If an error occurs,
C* attempt cleanup.
C*
C                   if        FileDesc = -1
C                   eval      PrintLine = 'Open of file failed'
C                   exsr      error
C                   eval      ReturnInt = unlink(%ADDR(FileName))
C                   eval      *INLR = '1'
C                   return
C                   else
C                   eval      PrintLine = 'Open of file successful'
```

```
C                   except
C                   endif
C*
C* Read from file
C*
C                   eval      BytesRead = read(FileDesc
C                               : %ADDR(Buffer) : %SIZE(Buffer))
C*
C* Check for error and report status.  If an error occurs,
C* attempt cleanup.
C*
C                   if        BytesRead = -1
C                   eval      PrintLine = 'Read failed'
C                   exsr      error
C                   eval      ReturnInt = close(FileDesc)
C                   eval      ReturnInt = unlink(%ADDR(FileName))
C                   eval      *INLR = '1'
C                   return
C                   else
C                   if        BytesRead = BytesWrt
C                   eval      PrintLine = 'Data successfully read: '
C                               + %TRIMR(Buffer)
C                   else
C                   eval      PrintLine = 'Data truncation on read'
C                   endif
C                   except
C                   endif
C*
C* Close the LinkName file
C*
C                   eval      ReturnInt = close(FileDesc)
C*
C* Check for error and report status.  If an error occurs,
C* attempt cleanup.
C*
C                   if        ReturnInt = -1
C                   eval      PrintLine = 'Close of link failed'
C                   exsr      error
C                   eval      ReturnInt = close(FileDesc)
C                   eval      ReturnInt = unlink(%ADDR(FileName))
C                   eval      *INLR = '1'
C                   return
C                   else
C                   eval      PrintLine = 'Close of link successful'
C                   except
C                   endif
C*
C* Unlink test_file
C*
C                   eval      ReturnInt = unlink(%ADDR(FileName))
C*
C* Check for error and report status.  If an error occurs,
C* attempt cleanup.
C*
C                   if        ReturnInt = -1
C                   eval      PrintLine = 'Unlink of file failed'
C                   exsr      error
C                   eval      ReturnInt = unlink(%ADDR(FileName))
C                   eval      *INLR = '1'
C                   return
C                   else
C                   eval      PrintLine = 'Unlink of file successful'
C                   except
C                   endif
C*
C* End of main program
C*
```

```
C                   eval      PrintLine = 'Program run is successful'
C                   except
C                   eval      *INLR = '1'
C                   return
C*
C* Common error reporting subroutine
C*
C* If errors occur, the integrated file system exports the variable
C* 'errno' to assist in determining the problem.  As 'errno' is
C* lowercase, ILE RPG cannot directly import this variable and must
C* use a C module to access it.  If the developer has ILE C
C* available, the following sample C code will import 'errno' and
C* make it available to the RPG application.
C*
C*     #include <errno.h>
C*     int geterrno()
C*     {
C*       return errno;
C*     }
C*
C* To activate this C module, remove the four comment identifiers
C* following the 'except' statement and remove the comment identifier
C* from the errno prototype.  Definitions for the returned errno
C* are found in the file QSYSINC/SYS member ERRNO.
C*
C     error         begsr
C                   except
C*                  eval      ReturnInt = errno
C*                  move      ReturnInt     Errnoval          9
C*                  eval      PrintLine = 'Errno is ' + Errnoval
C*                  except
C                   eval      PrintLine = 'Program ended in error'
C                   except
C                   endsr
OQSYSPRT   E                            1
O                       PrintLine          100
```

API information format

The format of the IBM i API information includes sections such as parameters, authorities and locks, required parameter group, format, field descriptions, and error messages.

Related concepts:

"APIs for the program-based environment" on page 10
Program-based APIs are called as programs (*PGMs). They are the initial APIs on the system.

"APIs for the service-program-based environment" on page 10
APIs based on service programs are called as procedures exported from ILE service programs (*SRVPGM).

"Receiver variables" on page 73
A *receiver variable* is a program variable that is used as an output field to contain information that is returned from a retrieve API.

Related reference:

"API naming conventions" on page 6
Program-based APIs and service-program-based APIs follow similar naming conventions.

API description

For most APIs, the description information contains similar sections.

- "Parameters" on page 48
- "Authorities and locks" on page 48
- "Required parameter group" on page 49
- "Optional parameter group" on page 50

This topic uses the Retrieve Job Description Information (QWDRJOBD) API as an example to illustrate how to use the information in each section. In the following discussion, assume that you are interested in accessing the value of the HOLD parameter of a job description. For example programs to which the discussion of the QWDRJOBD API is related, see "Examples: Program-based APIs" on page 120.

Parameters

The Parameters box describes how to call the API.

The first column in the Parameters box lists the required order of the parameters.

The second column lists each parameter used on the call.

The third column lists whether the parameter is defined for input, output, or input and output. Input parameters and fields are not changed by the API. They have the same values on the return from the API call as they do before the API call. In contrast, output parameters are changed. Any information that an API caller places in an output parameter or output field before the call can be lost on the return from the call.

The fourth column of the Parameters box lists the type of data that is defined for the parameter. CHAR(*) represents a data type that is indeterminate, such as a data structure, or represents a data type of a length that is not fixed. Binary(*x*) represents *x* bytes of a binary value. CHAR(*x*) represents *x* bytes of character data. For example, the Retrieve Job Description Information (QWDRJOBD) API has parameters such as an 8-byte character format name, a 4-byte binary value named length of receiver variable, and a variable-length receiver variable. The receiver variable is a structure that consists of several character and binary fields. For more information about format names, see Format name.

Example: RPG call statement parameters

For the QWDRJOBD API, you must pass 5 parameters as shown in the following RPG CALL statement:

```
C                   CALL 'QWDRJOBD'
C                   PARM           QWDBH           Receiver Var.
C                   PARM           RCVLEN          Length QWDBH
C                   PARM           FORMAT          Format Name
C                   PARM           LFNAM           Qual. Job Desc
C                   PARM           QUSBN           Error Code
```

Note: There is no parameter for the HOLD information. The first parameter, receiver variable (QWDBH), is where the information is passed back from the QWDRJOBD API. You receive a data structure that contains the information. You need to find the specific location within the data structure for where the HOLD information is stored.

For complete RPG example programs that use the QWDRJOBD API, see "Examples: Program-based APIs" on page 120.

Authorities and locks

The Authorities and Locks section lists all the authorities that you need to use an API. This section also lists the locks that the API uses.

To use an API, you must have the correct authority to the API, to all the objects that the API uses, and to any locks that the API places on any objects.

Locks are based on the objects that the API uses. The type of locking that occurs, such as whether the object can be used by more than one user at the same time, is based on what actions the API performs on the object.

For the QWDRJOBD API, you must have *USE authority to both the job description object and the library to access the object. This is the same type of authority that is required for most situations where you want to display or retrieve information in an object. For example, it is the same authority that you would need to use the Display Job Description (DSPJOBD) command. Because no specific information is described for locks, you can assume that nothing unusual is required.

Required parameter group

The Required Parameter Group section lists all the parameters that are required for an API. You must use all of the parameters in the order that they are listed. None of the parameters can be left out.

The details about each parameter that must be used on the call to the QWDRJOBD API are described in the Required Parameter Group section in Retrieve Job Description Information (QWDRJOBD) API.

Receiver variable

A receiver variable is the name of the variable where the retrieved information is placed, for example, QWDBH in the example RPG program in "Parameters" on page 48.

You need to declare the length of the receiver variable based on what you want from the format. The include file QWDRJOBD contains the definition for the receiver variable structure depending on the value used for the format name. For more information about the format, see the table in JOBD0100 Format.

You can see from the *Dec* (decimal offset) column of the JOBD0100 format table that at least 390 bytes plus additional bytes (of unknown length) for the initial library list and the request data are returned. "Example in OPM RPG: Accessing a field value (initial library list)" on page 153 describes how to determine the lengths of these fields. For now, you should focus on the fixed portion (390 bytes) of the format.

You can receive the maximum or enough bytes to contain the information in which you are interested. Because the value of the hold on job queue field starts at decimal 76, you can specify that the receiver variable is 100 bytes in length (or any number greater than or equal to 86 bytes). You do not need to be precise about the length of the receiver variable. Whatever you specify is the amount of data that is returned. You can truncate a value in the middle of a field in the format or specify more length than the format has.

Assume that you want to receive the fixed information, a length of 390 bytes, shown at (1) in "Example in OPM RPG: Retrieving the HOLD parameter (exception message)" on page 121. If you are going to call the API once, no measurable performance gain occurs if you specify anything less than the maximum. When defining the length of your receiver variable, you usually use the length of the information that you want to receive. The length of the receiver variable parameter must be set to a value equal to or less than the length that you defined the receiver variable parameter to be.

Length of receiver variable

You normally enter the length that you have specified for the receiver variable. Remember that in this example, you want to declare the receiver variable to be 390 bytes in length. The length of the receiver variable parameter has a value of 390 assigned to it, shown at (2) in "Example in OPM RPG: Retrieving the HOLD parameter (exception message)" on page 121. You could have specified a different value, but the value must be the same or less than the size of the variable in your program. In Example: RPG call statement parameters, RCVLEN is the length of the receiver variable parameter.

The length field, according to the required parameter group, must be described as BINARY(4). This means that a field of 4 bytes is passed where the value is specified in binary. You need to know how your high-level language allows you to define a 4-byte field and place a binary value in it.

Format name

A format name is a name that identifies what type of information you want returned in the receiver variable. Because the QWDRJOBD API has a single format name, JOBD0100, you use this format name shown at (3) in "Example in OPM RPG: Retrieving the HOLD parameter (exception message)" on page 121. The format name variable in the example program is called FORMAT. You can place the format name in a variable or pass it as a literal.

Qualified job description name

This name must be passed as a 20-character name with the job description name in the first 10 characters and the library qualifier beginning in the 11th character. If you want JOBD1 in LIBX, specify as follows:

```
1         11        20
.          .         .
.          .         .
JOBD1      LIBX
```

The special value *CURLIB or *LIBL can be used as the library qualifier.

Note: APIs generally do not convert parameter values to uppercase. When using object names (like job description and library), you must provide the names in uppercase.

Error code

This parameter allows you to select how errors are to be handled.

The include file QUSEC contains the definition for the error code structure that is used for the error code parameter.

You can choose to receive exceptions (escape messages) or to receive an error-code data structure that allows you to determine whether an exception occurs. Depending on your high-level language, you might not have a choice for which method you use. You might have to use the error-code data structure because some languages do not provide for the monitoring of escape messages.

In "Example in OPM RPG: Retrieving the HOLD parameter (exception message)" on page 121, the RPG program requests that exceptions be sent if any errors occur. To provide this type of exception handling, a 4-byte binary field with a value of zero must be passed, as shown at (4). This indicates to the API that you want exception messages to be sent.

Optional parameter group

Some APIs have optional parameters. The optional parameters form a group. You must either include or exclude the entire group. You cannot use only one of these parameters. You must include all preceding parameters.

The API can be called either with or without the optional parameters.

The Retrieve Job Description Information (QWDRJOBD) API does not have an optional parameter group. The List Job (QUSLJOB) API has an optional parameter group.

Related reference:

List Job (QUSLJOB) API

API format

The format section in the API information shows the type of information to be returned by an API.

For example, for the Retrieve Job Description Information (QWDRJOBD) API, the format described is JOBD0100 Format. Listed within the format are the individual fields that contain the attributes of the job description. The offset in the Dec (decimal offset) column for the hold on job queue field (hold parameter on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command) begins at decimal offset 76.

The fields in the format do not occur in any particular sequence. To determine what you want, you need to scan the format.

The QWDRJOBD API has only a single format; other APIs can have multiple formats where each format has different levels of information. With multiple formats, a format name parameter allows you to specify which format you want to retrieve.

Related reference:

Retrieve Job Description Information (QWDRJOBD) API

"General data structure for list APIs" on page 76
The data structure for the list APIs consists of several fields.

API field descriptions

The field descriptions section in the API information describes the fields in each API format.

The contents of the format are presented in alphabetic sequence, not in the sequence of the fields defined in the format. In the Retrieve Job Description Information (QWDRJOBD) API, you can find the description of the hold on job queue field. The field does not use the parameter name found on the Create Job Description (CRTJOBD) command.

Related reference:

Retrieve Job Description Information (QWDRJOBD) API

API error messages

The error messages section in the API information lists the error messages for each API.

Message IDs normally exist in the QCPFMSG file. You might want to program for these messages, regardless of the high-level language that you use. If you need more information about the messages, use the Display Message Description (DSPMSGD) command.

Related reference:

Retrieve Job Description Information (QWDRJOBD) API

Display Message Description (DSPMSGD) command

Extracting a field from the format

You can determine from the API format section where the field that you want to extract is located within the receiver variable.

An offset is shown in both decimal and hexadecimal. Depending on the high-level language that you use, either offset might be helpful. For CL and RPG, you normally use the decimal offset. With either offset, you must remember whether your language works with the offset from a base of 0 or a base of 1. The API format tables are prepared for languages that work from a base of 0, but not all languages can use this base. CL and RPG, for example, work from a base of 1, so you need to add 1 to the decimal value of each offset. The hold on job queue field begins at decimal offset 76, for example. To access the information in CL or RPG, you need to address byte 77 within the receiver variable.

Using the format, you can tell that the field after the hold on job queue field, output queue name, begins in offset 86. This means that the hold on job queue information is in the following location from a CL or RPG perspective:

```
77        86
.         .
.         .
XXXXXXXXXX
```

The only possible values for the hold on job queue field are *YES and *NO. They are left-aligned in the field and the remaining positions are blank.

Most of the formats provide additional bytes for each field to allow expansion, such as a new value for the hold on job queue field that is more than 4 bytes.

Many of the needed structures are provided by the system include library, QSYSINC. However, any fields of a structure that are variable in length are not defined in the QSYSINC library. These variable-length fields must be defined by the user, as shown at (3) in "Example in OPM RPG: Accessing a field value (initial library list)" on page 153.

Related concepts:

"Include files and the QSYSINC library" on page 59
An *Include file* is a text file that contains declarations that are used by a group of functions, programs, or users. The system include (QSYSINC) library provides all source include files for APIs that are included with the IBM i operating system.

Related reference:

"General data structure for list APIs" on page 76
The data structure for the list APIs consists of several fields.

Processing lists that contain data structures

Some API formats contain a list where each entry in the list is a data structure.

A good example is the Retrieve System Status (QWCRSSTS) API. It supports multiple formats for different types of information. The SSTS0300 format contains a list where each entry in the list has the information about a particular storage pool. In addition to the two critical fields (the offset to where the list begins field and the number of entries in the list field), the format also supports a field that describes the length of each entry. In the initial library list, each entry was 11 bytes long. But in a storage pool, a field (length of pool information entry) describes the length and should be used instead of a fixed-length increment. This allows for growth, such as more information being available in another release for each list entry.

If another field is added to describe additional information about a storage pool, it is probably added after the paging option field. The length of pool information entry allows your code to be compatible with future releases while it retains the locations (relative to the start of a list entry) of the current fields.

Related reference:

Retrieve System Status (QWCRSSTS) API

API parameters

After you decide which API to use, you need to code a call to the API and pass to the API the set of parameters that are appropriate for it.

There are three types of parameters:

- *Required*: All of the parameters are in the specified order.
- *Optional*: All or none of the parameters are within the optional group. You must either include or exclude the entire group. You cannot use only one of these parameters. In addition, you must include all preceding parameters.
- *Omissible*: The parameters can be omitted. When these parameters are omitted, you must pass a null pointer.

For program-based and service-program-based APIs, the values for all parameters that identify objects on the system must be in *NAME (basic name) format, left-aligned, uppercase, and with valid special characters. (The *NAME format is a character string that must begin with an alphabetic character (A through Z, $, #, or @) followed by up to 9 characters (A through Z, 0 through 9, $, #, @,), or _). The system uses an object name as is; it does not change or check the object name before locating the object. This can improve the performance of the API. An incorrect name usually causes an `Object not found` error.

Related reference:

"Differences between program-based APIs and service-program-based APIs" on page 12
Program-based APIs and service-program-based APIs are different in API names, parameters, error conditions, and pointer use.

Passing parameters

High-level languages pass parameters to an API by value, directly; by value, indirectly; or by reference.

Depending on the high-level language that you use, parameters can be passed in the following ways:

By value, directly

The value of the data object is placed directly into the parameter list.

By value, indirectly

The value of the data object is copied to a temporary location. The address of the copy (a pointer) is placed into the parameter list. By value, indirectly is not done explicitly by the application programmer. It is done by the high-level language at run time.

By reference

A pointer to the data object is placed into the parameter list. Changes made by the called API to the parameter are reflected in the calling application.

When you call an API, the protocol for passing parameters is to typically pass a space pointer that points to the information being passed. (This is also referred to as pass-by-reference.) This is the convention used by default for the control language (CL), RPG, and COBOL compilers. Care must be used in those languages that support pass-by-value (such as ILE C) to ensure that these conventions are followed. Refer to the appropriate language documentation for instructions. The parameter passing convention of pass-by-reference can be used in all programming languages. Some of the UNIX-type APIs require pass-by-value parameter passing. VisualAge C++ for IBM i also supports pass-by-value parameter passing.

High-level semantics usually determine when data is passed by value and when it is passed by reference. For example, ILE C passes and accepts parameters by value, directly, while for OPM and ILE COBOL and OPM and ILE RPG parameters are usually passed by reference. You must ensure that the calling program or procedure passes parameters in the manner expected by the called API. The OPM or ILE HLL programmer's guides contain more information about passing parameters to different languages.

The ILE languages support the following parameter-passing styles:

- ILE C passes and accepts parameters by value (directly and indirectly) and by reference.
- ILE COBOL supports the passing of parameters by value (directly and indirectly) and by reference.
- ILE RPG supports the passing of parameters by value (directly and indirectly) and by reference.
- ILE CL supports the passing of parameters by value (directly) and by reference.

Input and output parameters

API parameters can be used for input or output. Some parameters, identified as input/output (I/O) parameters, contain both input and output fields.

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

Parameters can be classified into the following general categories:

- *Input parameters*: You must set input parameters before calling an API because these parameters pass needed information to the API to enable it to perform its function. For example, if the API is to perform a function on an object, one of the parameters would be the name and library of the object. Input parameters are not changed by the API.
- *Output parameters*: You do not need to set output parameters before calling an API because the API returns information to the application in these parameters. When a return to the application is successful and no errors occur, the application accesses the information returned in output parameters.
- *I/O parameters*: I/O parameters are identified as structures that contain fields. The fields within the structure can be either input, output, or both. For example, the bytes provided field in the error code parameter is an input field. The rest of the fields that comprise this parameter are output fields. The rules for input parameters and output parameters apply to the individual fields in the structure.

Offset values and lengths

When you use an API that generates a list of information into a user space, use the offset values and lengths returned by the API in the generic header of the user space to step through the list.

Because of the following considerations, you need to use the offset values and lengths returned by a list API in the generic header of the user space, rather than specifying what the current version of the API returns:

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

As long as your high-level language application program uses the offset values and lengths returned in the generic header of the user space, your program will run in future releases of the IBM i licensed program.

Note: While your application program should use the length returned in the generic header to address subsequent list entries, your application program should retrieve only as many bytes as it has allocated storage for.

Offset versus displacement considerations for structures

Some APIs use the term *offset* while other APIs use the term *displacement* to describe distances.

For example, the Retrieve Data Queue Message (QMHRDQM) API uses offset; the List Objects (QUSLOBJ) API uses displacement.

An *offset* is the distance from the beginning of an object (user spaces and receiver variables) to the beginning of a particular field. However, a *displacement* is the distance from the beginning of a specific record, block, or structure to the beginning of a particular field.

Error code parameter

An API error code parameter is a variable-length structure that is common to most of the system APIs. The error code parameter controls how errors are returned to the program.

The error code parameter must be initialized before the program calls the API. Depending on how the error code structure is set, this parameter either returns information associated with an error condition or causes errors to be returned as exception messages.

For some APIs, the error code parameter is optional. If you do not code the optional error code parameter, the API returns diagnostic and escape messages. If you code the optional error code parameter, the API can either signal exceptions or return the exception information in the error code parameter.

Notes:

- The ILE CEE APIs use feedback codes and conditions.
- The UNIX-type APIs use *errnos* to report error conditions.

The error code structure is provided in the QSYSINC library and is called QUSEC.

Related concepts:

"Include files and the QSYSINC library" on page 59
An *Include file* is a text file that contains declarations that are used by a group of functions, programs, or users. The system include (QSYSINC) library provides all source include files for APIs that are included with the IBM i operating system.

Related reference:

Errno Values for UNIX-Type Functions

IBM i Messages and the ILE CEE API Feedback Code

Error code parameter format:

Most IBM i APIs include an error code parameter to return error codes and exception data to the application.

The error code parameter can be one of these variable-length structures: format ERRC0100 or format ERRC0200.

In format ERRC0100, one field is an INPUT field; it controls whether an exception is returned to the application or the error code structure is filled in with the exception information. When the bytes provided field is greater than or equal to 8, the rest of the error code structure is filled in with the OUTPUT exception information associated with the error. When the bytes provided INPUT field is zero, all other fields are ignored and an exception is returned.

Format ERRC0200 must be used if the API caller wants convertible character (CCHAR) support. Format ERRC0200 contains two INPUT fields. The first field, called the key field, must contain a -1 to use CCHAR support. When the bytes provided field is greater than or equal to 12, the rest of the error code structure is filled in with the OUTPUT exception information associated with the error. When the bytes provided INPUT field is zero, all other fields are ignored and an exception is returned.

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

The following tables show the structures of the error code parameter. The fields are described in detail after the tables.

Note: The error code structures for both formats are provided in the QUSEC include file in the QSYSINC library. Include files exist in these source physical files: QRPGSRC, QRPGLESRC, QLBLSRC, QCBLLESRC, and H.

Format ERRC0100

Table 5. Format ERRC0100 for the error code parameter

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

Format ERRC0200

Table 6. Format ERRC0200 for the error code parameter

Offset		Use	Type	Field
Dec	**Hex**			
0	0	INPUT	BINARY(4)	Key
4	4	INPUT	BINARY(4)	Bytes provided
8	8	OUTPUT	BINARY(4)	Bytes available
12	C	OUTPUT	CHAR(7)	Exception ID
19	13	OUTPUT	CHAR(1)	Reserved
20	14	OUTPUT	BINARY(4)	CCSID of the CCHAR data
24	18	OUTPUT	BINARY(4)	Offset to the exception data
28	1C	OUTPUT	BINARY(4)	Length of the exception data
		OUTPUT	CHAR(*)	Exception data

Field descriptions

Fields in the error code structures are described in alphabetic order.

Bytes available. The length of the error information available for the API to return, in bytes. If it is 0, no error was detected and none of the fields that follow this field in the structure are changed.

Bytes provided. The number of bytes that the calling application provides for the error code. If the API caller is using format ERRC0100, the bytes provided must be 0, 8, or more than 8. If more than 32 783 bytes (32KB for exception data plus 16 bytes for other fields) are specified, it is not an error, but only 32 767 bytes (32KB) can be returned in the exception data.

If the API caller is using format ERRC0200, the bytes provided must be 0, 12, or more than 12. If more than 32 799 bytes (32KB for exception data plus 32 bytes for other fields) are specified, it is not an error, but only 32 767 bytes (32KB) can be returned in the exception data.

Table 7. Possible values for bytes provided

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

Table 7. Possible values for bytes provided (continued)

Bytes	Description
>=8	If an error occurs, the space is filled in with the exception information. No exception is returned. This occurs only for format ERRC0100.
>=12	If an error occurs, the space is filled in with the exception information. No exception is returned. This occurs for formats ERRC0100 and ERRC0200.

CCSID of the CCHAR data. The coded character set identifier (CCSID) of the convertible character (CCHAR) portion of the exception data.

Table 8. Possible values for CCSID of the CCHAR data

CCSID	Description
0	The default job CCSID.
CCSID	A valid CCSID number. The valid CCSID range is 1 through 65535, but not 65534.

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

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

Key. The key value that enables the message handler error function if CCHAR support is used. This value should be -1 if CCHAR support is expected.

Length of the exception data. The length, in bytes, of the exception data returned in the error code.

Offset to the exception data. The offset from the beginning of the error code structure to the exception data in the error code structure.

Reserved. A 1-byte reserved field.

Examples: Receiving error conditions:

Depending on how you set the error code parameter, your application can receive error conditions as exceptions or receive the error code with or without exception data.

Example: Receiving error conditions as exceptions

This example defines an error code structure that receives error conditions as exceptions using format ERRC0100 of the error code structure. The application allocates an error code parameter that is a minimum of 4 bytes long to hold the bytes provided field. The only field used is the bytes provided INPUT field, which the application sets to zero to request exceptions. The error code parameter contains the following field.

Table 9. Error code structure for receiving error conditions as exceptions

Field	INPUT	OUTPUT
Bytes provided	0	0

Example: Receiving the error code without the exception data

This example defines a format ERRC0100 error code structure that receives the error message or exception ID but no exception replacement data. To do this, the application allocates an error code parameter that is a minimum of 16 bytes long for the bytes provided, bytes available, exception ID, and reserved fields. It sets the bytes provided field of the error code parameter to 16.

If the called API were to return the error message CPF7B03, the error code parameter would contain the data shown in the following table. In this example, 16 bytes are provided for data, but 36 bytes are available. Twenty more bytes of data could be returned if the bytes provided field were set to reflect a larger error code parameter.

Table 10. Error code structure for receiving the error code without the exception data

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

Example: Receiving the error code with the exception data

This example defines a format ERRC0100 error code structure that receives the error message or exception ID and up to 100 bytes of exception replacement data that is associated with the exception. To do this, the application allocates an error code parameter that is 116 bytes long: 16 bytes for the bytes provided, bytes available, exception ID, and reserved fields, and 100 bytes for the exception data for the exception. (In some cases, the exception data might be a variable-length directory or file name, so this might not be large enough to hold all of the data; whatever fits is returned in the error code parameter.) Finally, it sets the bytes provided field to 116.

If the called API were to return the error message CPF7B03 with replacement variable &1 set to the value 'USRMSG ' and replacement variable &2 set to the value 'QGPL ', the error code parameter would contain the data shown in the following table.

Table 11. Error code structure for receiving the error code with the exception data

Field	INPUT	OUTPUT
Bytes provided	116	116
Bytes available	Ignored	36
Exception ID	Ignored	CPF7B03
Reserved	Ignored	0
Exception data	Ignored	USRMSG QGPL

Using the job log to diagnose API errors:

When your program encounters an API error, use messages in the job log to determine the cause.

Sometimes an API might issue a message stating that the API failed, and the message might direct you to see the previously listed messages in the job log. If you need to determine the cause of an error message, use the Receive Message (RCVMSG) command or the receive message APIs to receive the messages that explain the reason for the error.

In some cases, you can write an application program to use the diagnostic message to identify and correct the parameter values that caused the error.

The following CL program receives error messages from the job log.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

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

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

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

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

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

             GOTO       CMDLBL(ALLDONE)

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

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

Include files and the QSYSINC library

An *Include file* is a text file that contains declarations that are used by a group of functions, programs, or users. The system include (QSYSINC) library provides all source include files for APIs that are included with the IBM i operating system.

The QSYSINC library is optionally installed. It is fully supported, which means that you can write an authorized program analysis report (APAR) if you find an error in an include file.

You can install the QSYSINC library by using the GO LICPGM functions of the IBM i operating system. Select the Install Licensed Programs option on the Work with Licensed Programs display and the IBM i System Openness Includes option on the Install Licensed Programs display.

The terms *include file* and *header file* are interchangeable and pertain to the contents of the QSYSINC library. These files are intended to be compatible with future releases.

The naming conventions for the include files are the same as either the API program name or the ILE service program name. If both exist, the include file has both names.

The following table lists the include files that are shipped with the QSYSINC library.

Table 12. Include files shipped with the QSYSINC library

Operating environment	Language	File name	Member name (header file)
Program-based APIs	ILE C [1]	H	API program name
	RPG	QRPGSRC	API program name or API program name with the letter E replacing the initial letter Q for members that contain array definitions
	ILE RPG	QRPGLESRC	API program name
	COBOL	QLBLSRC	API program name
	ILE COBOL	QCBLLESRC	API program name
Service-program-based APIs	ILE C	H	Service program name or API program name [2]
	ILE RPG	QRPGLESRC	Service program name or API program name [2]
	ILE COBOL	QCBLLESRC	Service program name or API program name [2]
UNIX-type APIs	ILE C	ARPA	Industry defined
	ILE C	H	Industry defined
	ILE C	NET	Industry defined
	ILE C	NETINET	Industry defined
	ILE C	NETNS	Industry defined
	ILE C	SYS	Industry defined

Notes:

1. ILE CEE APIs are included in this part of the table.
2. The API can be either bindable when you use the service program name or callable when you use the API program name.

Besides the include files for specific APIs, the QSYSINC library also contains the following include files.

Table 13. Other include files in the QSYSINC library

File name	Description
QLIEPT and QUSEPT	Allows C language application programs to call program-based APIs directly through the system entry point table.
QUSGEN	Defines the generic header for the list APIs.

Table 13. Other include files in the QSYSINC library (continued)

File name	Description
QUSEC	Contains the structures for the error code parameter.
Qxx	Provides common structures that are used by multiple APIs (where the *xx* is the component identifier, for example, QMH, QSY, and so forth).

The include files that are included with the system define only the fixed portions of the formats. You must define the varying-length fields. The QSYSINC include files are read-only files. If you use a structure that contains one or more varying-length fields, you have two options for defining these varying-length fields. You can copy the include file to your own source file and edit your copy. Uncomment the varying-length fields in your copy of the include file, and specify the actual lengths you want.

Alternatively, if you develop with an ILE language, you can reference the QSYSINC definitions using ILE RPG LIKEDS or ILE COBOL TYPEDEF support in order to define both the fixed- and varying-length portions of structures. When you use a structure as input to an API, initialize the structure in its entirety (typically to x'00', but refer to the specific API documentation for the correct value) before setting specific field values within the structure. This saves you from initializing reserved fields by name because the reserved field name might change in future releases.

An exit program has an include file only when it contains a structure. The member names for exit programs start with the letter E. Except for RPG array definitions for APIs that also start with E, any member names in the QSYSINC library that start with the letter E are include files for exit programs. The QSYSINC member name of these include files is provided in the parameter box for the applicable exit programs.

For development of client-based applications, integrated-file-system symbolic links to QSYSINC openness includes are also provided in the /QIBM/include path.

All source physical files are included with read capabilities only; changes cannot be made to the QSYSINC library. All these files are built with a CCSID of 00037. When you compile a program in a specific CCSID, any QSYSINC include file is converted to the program CCSID.

If you are coding in ILE C, the header files in the QSYSINC library are considered system include files. You should use the < and > symbols on the #include statement; this affects how the library list is used to search for header files.

If you are developing applications on a release *n* system that will run on a release *n-1* system, you might want to copy the include files of each release to user source libraries. This minimizes the impact of include file changes when APIs are enhanced over time with additional fields.

Related concepts:

"Extracting a field from the format" on page 51
You can determine from the API format section where the field that you want to extract is located within the receiver variable.

Related reference:

"Error code parameter" on page 54
An API error code parameter is a variable-length structure that is common to most of the system APIs. The error code parameter controls how errors are returned to the program.

"Examples: APIs and exit programs" on page 297
These examples show how to use a wide variety of APIs and exit programs.

Internal object types

Internal objects are used to store the information needed to perform some system functions. The table shows the predefined values for all the IBM i internal object types.

Table 14. Predefined values and default library locations for internal IBM i object types

Value	Object type	Hexadecimal format
*ACNAME	Auto-configuration names	19F0
*ADO	Asynchronous distribution object	19E0
*AUT	Authorized user table	0EC5
*AUTHLR	Authority holder	1BC1
*CBLK	Commit block	0FC1
*CCSIDI	CCSID information	0ED2
*CDJOBLK	Transaction control structure with externally-managed commitment definitions	23A1
*CDTCSLK	Transaction control structure with DB2-managed commitment definitions	23A0
*CFGSPC	Configuration space	19A4
*CHRSFC	Character special file clone object	1EC1
*CIO	Cluster information object	19A5
*CMTCDRI	Commit recovery object	0EA0
*CNVTBL	System conversion table	19FB
*CRGM	Cluster resource group manager	0EA5
*DBCOLES	Database column extension	1955
*DBDIR	Database directory	1950
*DBRCVR	Database recovery object	19D4
*DCRENO	DC rename object	19F5
*DCTQ	Data dictionary queue	0AC4
*DCXITC	DCX inter-task communication index	0ECF
*DCXMSQ	DCX operator message queue	0AC5
*DEACR	Directory extended attribute cursor	0D52
*DEADI	Directory extended attribute index	0C50
*DEADS	Directory extended attribute dataspace	0B51
*DFTJRN	Default journal	09C1
*DFTRCV	Default journal receiver	07C1
*DIRCR	Directory cursor	0D51
*DIRDS	Directory dataspace	0B50
*DIRJ	Integrated file system directory for user journaling	1E50
*DLSTMF	Document library services stream file	1EA0
*DMPSP	Dump space	1390
*DOCBSS	Document byte string space	06C1
*DRQ	Distribution recipient queue	0AC3

Table 14. Predefined values and default library locations for internal IBM i object types (continued)

Value	Object type	Hexadecimal format
*DRX	Distribution recipient index	0ED1
*DSNXO	DSNX system object	19E9
*DTO	Distribution tracking object	19E2
*DUO	Document unit object	19E3
*EDTIDX	Element description index	OED0
*EPTAB	Data management entry point table	19D7
*EXITSP	Exit registration space	1953
*FACB	File available control block	0EC6
*FCNUL	Function usage list	0ECA
*FIDTBL	File ID table	0BA0
*FCS	File constraint space	1958
*FMT	File format	1951
*FSO	FMS system object	19E8
*GDA	Group data area	19CD
*GENIDX	Permanent generic index	0EA4
*GENQ	Permanent generic queue	0AC8
*GRPDLS	Group dataLink space	1959
*HFSD	HFS description	19F7
*HPQ	Office host print queue	0AC6
*ICO	Install communication object	19C6
*IDDEDT	Internal data dictionary	19EB
*IGCINT	Ideographic character table	19E1
*IMPLREP	Implementation repository	0E50
*INAUT	Install authority object	19D5
*INAUTO	Automatic install	19F4
*INITSP	Install initial template space	19C1
*INTLIB	Internal library	04C1
*INTPRF	Interactive profile	0EC4
*IPLJMQ	LIC internally-created queue space	18A1
*IFSIDX	Integrated file system index	0EF3
*ISYSLIB	Internal system library	04C2
*JAR	Job APAR repository	19CA
*JVAGRP	Java™ group	2150
*JVAPGM	Java program	0250
*JMQ	Job message queue	18A0
*JRNIX	Journal receiver index	0EA6
*JSQ	Job schedule queue	0AF0
*JTMMQ	Measurement message queue	0AC1
*LDA	Local data area	19CE
*LIBRCVR	Library recovery object	19D1

Table 14. Predefined values and default library locations for internal IBM i object types (continued)

Value	Object type	Hexadecimal format
*LIRCVR	Library recovery object for rename	19F2
*MCBSF	Management Collection Byte Stream File	1E52
*MCO	Measurement collection object	19C0
*MCOTBL	Measurement collection object table	19C8
*MDO	Measurement descriptor object	19C9
*MDOC	Mail document	19E6
*MEM	database file member	0D50
*MNINX	Menu index	0EC1
*MNTXT	Menu text	19CB
*MQLOCK	Message queue locking protocol	19DF
*MSCSP	Permanent miscellaneous space	19EE
*MSRVI	master service index	0E91
*NFSP	Network facility space	19E5
*OCUR	Database operational cursor	0DEF
*OHCUR	Database operational hybrid cursor	0DEE
*OIRS	OIR space	1952
*OLBSF	Open List Byte Stream File	1E51
*OPTBSS	Optical byte string	06A0
*OPTSTMF	Optical stream file	1EED
*OSSCB	Session control block	19E4
*OWCUR	Database operational wrapper cursor	ODED
*PCCR	Problem change control record	19CC
*PDT	Process definition template	19C7
*POBSF	Print Object Byte Stream File	1EB2
*PRDAVLI	Product availability index	0EF1
*PRMGEN	Permanent generic space	19A1
*PROCT	Operation code table	19DA
*PRODT	Operand description table	0ECB
*PRTQ	Print queue	0EC7
*PTCSPC	Protected space	19FC
*QDAG	Access group	0190
*QDDS	Data space	0B90
*QDDSI	Data space index	0C90
*QDIDX	Composite index	0E90
*QDPCS	Process control space	1A90
*QDQ	Composite queue	0A90
*QDSP	Composite space	1990
*QFSIDX	QSYS directory I/O index	0EA3
*QTAG	Temporary access group	01EF

Table 14. Predefined values and default library locations for internal IBM i object types (continued)

Value	Object type	Hexadecimal format
*QTDS	Temporary data space	0BEF
*QTDSI	Temporary data space index	0CEF
*QTIDX	Temporary index	0EEF
*QTPCS	Temporary process control space	1AEF
*QTQ	Temporary queue	0AEF
*QTSP	Temporary space	19EF
*RCYAP	Recovery times for SMAPP	19A0
*RWCB	Read/write control block	19C5
*RZHRIPD	HRI-saved persistent data	19A3
*SCO	Service communication object	19DC
*SCPFSP	SCPF space	19DE
*SDQ	SNADS distribution queue	0EC2
*SECOBJ	Internal security object for a user profile	0EC3
*SEPT	System entry point table	19C3
*SHRCV	SH recovery object	19F8
*SIQ	FM queue	0AC2
*SLFSMS	Secondary logic unit index	0EC9
*SMBSF	Shared memory byte stream file	1EB1
*SMIDX	System management index	0EF2
*SMQ	System management internal queue	0AF1
*SNMTBL	System program name table	19F9
*SORTSEQ	Sort sequence table repository	0EA7
*SPLCB	Spool control block	19C2
*SRAUTH	Save/restore authorizations table	19CF
*SRDS	Save/restore descriptor space	19DB
*SRMIDX	System resource manager index	0EC8
*SRMSPC	System resource manager space	19F3
*STPWIDX	Password index	0ED3
*STREAM	Stream object	85A0
*SVAL	System value	19D2
*SVRSTGD	Server storage space	1954
*SWFL	System-wide folder list	0ECD
*SYAUTS	Security authorization space	19F6
*SYSBC	System control block	19D3
*SYSPRTI	System print image part numbers	19D6
*SYSRPYL	System reply list	19D8
*S36BCH	System/36 batch object	19F1
*S36EPT	System/36 entry point table	19EA
*S36HLP	System/36 help object	0ECE

Table 14. Predefined values and default library locations for internal IBM i object types (continued)

Value	Object type	Hexadecimal format
*S36HST	System/36 history object	19EC
*S36IDX	System/36 index	0ECC
*TCPIPQ	TCP/IP queue	0AC7
*TDS	Trigger definition space	1960
*TNIPLMQ	TN IPL message queue	0AF2
*TOKTBL	DSOM token mapping table	0EA2
*UBPSPC	Usage-based pricing space	19FE
*UFCB	User file control block	19D9
*UFO	Unfiled folder object	19E7
*WCBT	Work control block table	19D0
*WCBTRO	Work control block table recovery object	19DD
*X40	X400 object	19FA
*X4Q	X400 queue	0EF0
*ZMFINX	Message space pool	0EA1
*ZMFSPC	Message framework	19A2

Related reference:

External object types

"Creating an MI version of the CLCRTPG program" on page 507
This topic discusses how to create an MI version of the CLCRTPG program that can be used to create MI programs. This program is called MICRTPG.

Data types and APIs

APIs support character data and binary data.

Character data

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

- A type that is not known, such as character or binary
- A length that might not be known or is based on another value (for example, a length you specify)

Binary data

In the API parameter tables, BINARY(2), BINARY(4), and BINARY(8) represent numeric data. These parameters must be signed, 2-, 4-, or 8-byte numeric values with a precision of 15 (halfword), 31 (fullword), or 43 bits and 1 high-order bit for the sign. Numeric parameters that must be unsigned numeric values are explicitly defined as BINARY(*x*) UNSIGNED.

When you develop applications that use binary values, be aware that some high-level languages allow the definition of binary variables by using precision and not length. For example, an RPG definition of binary length 4 specifies a precision of 4 digits, which can be stored in a 2-byte binary field. For API BINARY fields, RPG developers should use one of the following:

- Positional notation (1 2B 0) for 2-byte binary
- Positional notation (1 4B 0) for 4-byte binary
- A length of 1 to 4 for 2-byte binary (4B 0)

- A length of 5 to 9 in order to allocate a 4-byte binary field (9B 0)
- A length of 5 for 2-byte signed integer (5i 0)
- A length of 5 for 2-byte unsigned integer (5u 0)
- A length of 10 to allocate a 4-byte signed integer field (10i 0)
- A length of 10 to allocate a 4-byte unsigned integer field (10u 0)
- A length of 20 to allocate an 8-byte signed integer field (20i 0)
- A length of 20 to allocate an 8-byte unsigned integer field (20u 0)

Related reference:

"Example in OPM RPG: Retrieving the HOLD parameter (error code structure)" on page 133
This OPM RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

"Defining data structures" on page 532
When a data structure is defined for use with an API, the structure must be built to receive what the API returns. Here are the program examples that show the incorrect and correct ways of defining data structures. You can prevent errors by using IBM-supplied data structures rather than creating your own data structures.

Internal identifiers

Several APIs either require or allow you to use an internal identifier (ID) to identify an external name. When you use an internal ID, the processing can be faster because the system does not need to convert the external name to the internal ID.

A variety of terminology is used to identify an internal ID. Here are some examples:

- Work management uses an *internal job identifier*.
- Spooling uses an *internal spooled file identifier*.
- Security uses the term *handle* to mean the user profile that is currently running the job.
- Message handling uses the term *message key* (also appears in CL commands) to identify a message in a message queue.

The internal values are often accessed in one API and then used in another. For example, if you want a list of jobs, you use the List Jobs (QUSLJOB) API, which provides the internal job ID for each job in the list. You can use the internal job ID to access a spooled file for a job with the Retrieve Spooled File Attributes (QUSRSPLA) API.

User spaces and receiver variables

List APIs return information to *user spaces*, and retrieve APIs return information to *receiver variables*.

User spaces

List APIs return information to user spaces. A *user space* is an object consisting of a collection of bytes that can be used for storing any user-defined information.

Here are some of the advantages of using user spaces:

- User spaces can be automatically extended.
- User spaces can be shared across jobs.
- User spaces can exist across initial program loads (IPLs).

To provide a consistent design and use of the user space (*USRSPC) objects, the list APIs use a general data structure.

Related concepts:

"User spaces for list APIs" on page 79
List APIs require a user space to contain returned information.
"Receiver variables" on page 73
A *receiver variable* is a program variable that is used as an output field to contain information that is returned from a retrieve API.
"List APIs overview" on page 76
List APIs return a list of information. They use a common generic header to provide information such as the number of list entries and the size of a list entry. The content of the list is unique to each API.

General data structure:

List APIs use a general data structure. A *data structure* is an area of storage that defines the layout of the fields within the area.

The following figure shows the general data structure for the list APIs.

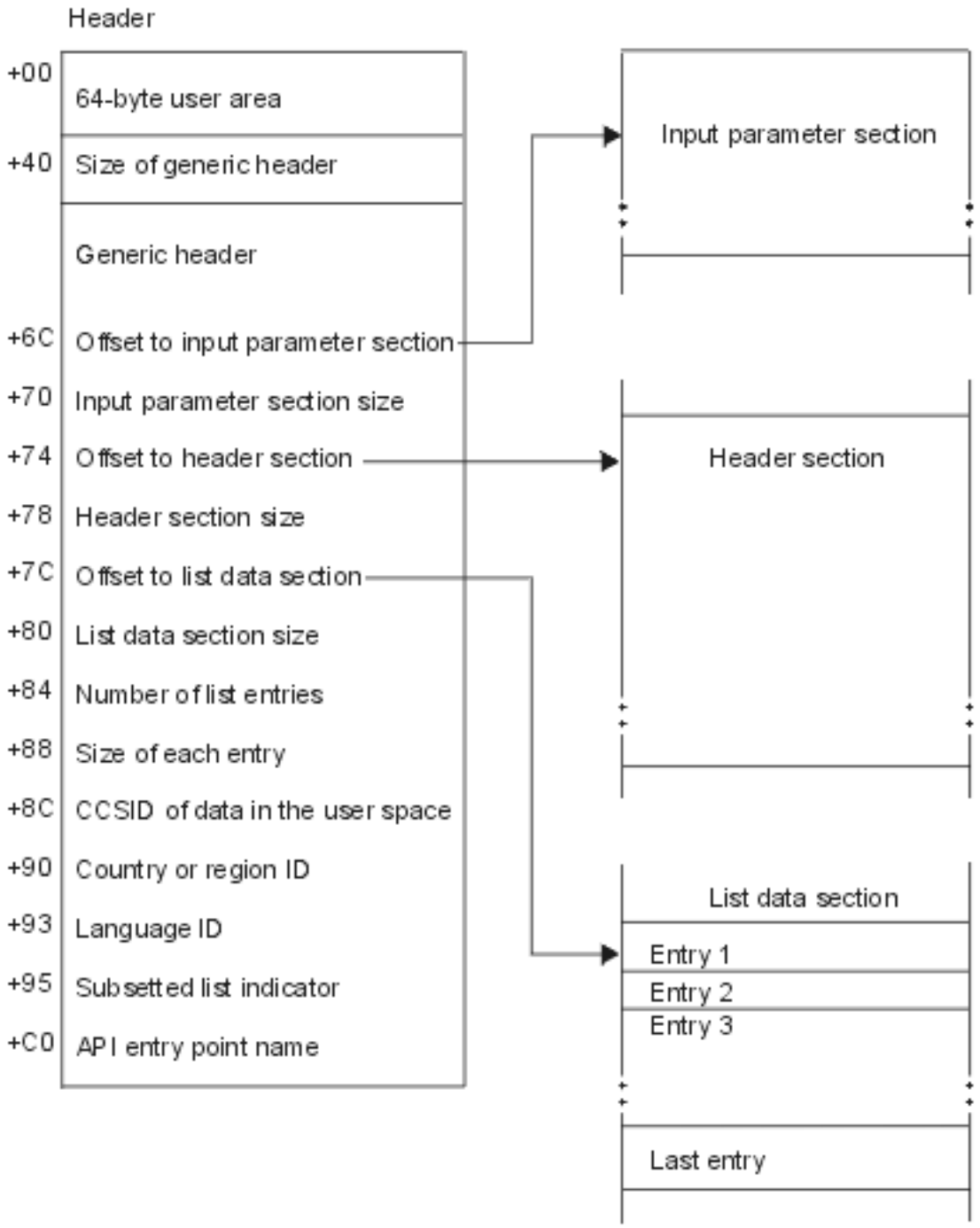


Figure 1. Data structure for list APIs

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

Common data structure formats:

Here are the data structure formats and field descriptions for the list APIs.

Generic header format 0100

Format 0100 is for the list APIs that are called as programs (*PGMs).

Offset		Type	Field
Dec	**Hex**		
0	0	CHAR(64)	User area
64	40	BINARY(4)	Size of generic header
68	44	CHAR(4)	Structure's release and level
72	48	CHAR(8)	Format name
80	50	CHAR(10)	API used
90	5A	CHAR(13)	Date and time created
103	67	CHAR(1)	Information status
104	68	BINARY(4)	Size of user space used
108	6C	BINARY(4)	Offset to input parameter section
112	70	BINARY(4)	Size of input parameter section
116	74	BINARY(4)	Offset to header section
120	78	BINARY(4)	Size of header section
124	7C	BINARY(4)	Offset to list data section
128	80	BINARY(4)	Size of list data section
132	84	BINARY(4)	Number of list entries
136	88	BINARY(4)	Size of each entry
140	8C	BINARY(4)	CCSID of data in the list entries
144	90	CHAR(2)	Country or region ID
146	92	CHAR(3)	Language ID
149	95	CHAR(1)	Subsetted list indicator
150	96	CHAR(42)	Reserved

Generic header format 0300

Format 0300 is for the list APIs that are called as procedures exported from ILE service programs (*SRVPGM).

Offset		Type	Field
Dec	**Hex**		
0	0		Everything from the 0100 format
192	C0	CHAR(256)	API entry point name
448	1C0	CHAR(128)	Reserved

Field descriptions

Fields in the data structure formats are described in alphabetic order.

API entry point name. The name of the ILE bindable API entry point that generates the list.

API used. For format 0100, this is the name of the program-based API that generates the list. For format 0300, this is a reserved field. See the API entry point name field for the API used.

CCSID of the data in the list entries. The coded character set ID for data in the list entries. If the value is 0, the data is not associated with a specific CCSID and should be treated as hexadecimal data.

Country or region ID. The country or region identifier of the data written to the user space.

Date and time created. The date and time when the list was created. The table shows the possible values.

Table 15. Possible values for date and time created

Value	Description
1	Century, where 0 indicates years 19 *xx* and 1 indicates years 20 *xx*.
2-7	The date, in YYMMDD (year, month, day) format.
8-13	The time of day, in HHMMSS (hours, minutes, seconds) format.

Format name. The name of the format for the list data section.

Information status. Whether the information is complete and accurate. The table shows the possible values.

Table 16. Possible values for information status

Value	Description
C	Complete and accurate.
I	Incomplete. The information that you receive is not accurate or complete.
P	Partial but accurate. The information that you receive is accurate, but the API has more information to return than the user space can hold. See “List sections” on page 73 for more information about partial lists.

Language ID. The language identifier of the data written to the user space.

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

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

Reserved. An ignored field.

Size of each entry. The size of each list data section entry, in bytes. All entries are the same size. For formats that return variable length records, this is zero.

Size of generic header. The size of the generic header, in bytes. This does not include the size of the user area. See “General data structure” on page 68 for a diagram showing the user area.

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

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

Size of list data section. The size of the list data section, in bytes. For formats that return variable length records, this is zero.

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

Structure's release and level. The release and level of the generic header format for this list. The value of this field is 0100 for generic header format 0100 and 0300 for generic header format 0300. List APIs put this value into the user space.

Subsetted list indicator. A flag that indicates whether the data selected from the list API can be stored in that format. The table shows the possible values.

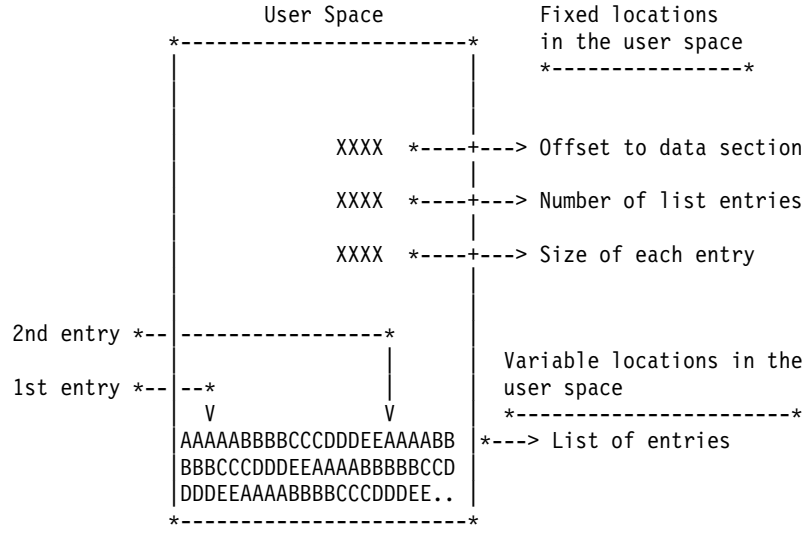
Table 17. Possible values for subsetted list indicator

Value	Description
0	List is not subsetted; all of the information can be stored in the format.
1	List is subsetted. For example, integrated file system names might be longer than the available area in the format.

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

Example: User space format:

This example illustrates the format of a user space. It does not contain all of the fields in the fixed portion of a user space.



List sections:

Each list API provides an input parameter section, a header section, and a list data section.

Table 18. List sections in a user space format

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

When you retrieve list entry information from a user space, use the allocated size defined in your application. To get the next entry, use the entry size returned in the generic header. The size of each entry might be padded at the end. If you do not use the entry size from the generic header, the result might not be valid.

Partial list considerations

Some APIs might be able to return more information to the application than fits in a receiver variable or a user space. The information returned is correct, but not complete.

If the list information is not complete, the first situation and possibly the second situation occur:

- A *P* is returned in the information status field of the generic user space layout.
- The API supports a continuation handle.

If an indicator of a partial list is returned and the API supports a continuation handle, the application should call the API again with the continuation handle in the list header section and specify that the list begins with the next entry to be returned.

Note: If this is the first time that the API attempts to return information, the continuation handle must be set to blanks. If the API does not support a continuation handle, you need to call the API again and to use more restrictive values for the parameters.

Related reference:

"General data structure" on page 68
List APIs use a general data structure. A *data structure* is an area of storage that defines the layout of the fields within the area.

Receiver variables

A *receiver variable* is a program variable that is used as an output field to contain information that is returned from a retrieve API.

Retrieve APIs use receiver variables rather than user spaces to place returned information. A retrieve API requires only addressability to storage of fixed size (typically a field or structure defined in your program). A list API, in comparison, requires a user space because the amount of information returned by a list API might be large and not of a predictable size.

Retrieve APIs that return information to receiver variables use the storage provided for the receiver variable parameter. The returned information is in a specific format. The format name is usually a parameter on the call to the API, and the format indicates to the API the information that you want returned. On the return from the call to the API, the caller parses through the receiver variable and

extracts the information that is needed. The caller knows how the information is returned by the documented format of the information. An API might have one or many formats that give you the flexibility to choose the information that you need.

Some formats have variable-length fields, some have only fixed-length fields, and some have repeating entries. To move through the information, some formats use offsets, some use lengths, and some use displacements. When the field is defined as an offset, the offset is always the number of bytes from the beginning of the receiver variable. When a length or displacement is used to move through the receiver variable entries, the length is always added to the current position within the receiver variable.

Offsets and displacements are not the same. An *offset* is relative to the beginning of a receiver variable or the beginning of a user space, whereas a *displacement* is relative to the current position of the pointer plus the value within the displacement field. If a format uses a displacement, you see the word *displacement* in the Field column of the API description.

Related concepts:

"API information format" on page 47
The format of the IBM i API information includes sections such as parameters, authorities and locks, required parameter group, format, field descriptions, and error messages.

Related reference:

"User spaces" on page 67
List APIs return information to user spaces. A *user space* is an object consisting of a collection of bytes that can be used for storing any user-defined information.

Example: Receiver variables using ILE APIs

Example: Keyed interface using ILE APIs

"Defining byte alignment" on page 546
Correct byte alignment ensures that an API reads the data from the beginning of a record rather than at some other point. Here are the program examples that show the incorrect and correct ways of defining byte alignment.

"Example in OPM RPG: Using keys with the List Spooled Files (QUSLSPL) API" on page 168
This OPM RPG program processes a list of spooled file information that you have specified using keys.

Bytes available and bytes returned fields:

Most formats used by retrieve APIs have a bytes available field and a bytes returned field.

The bytes available field contains the length in bytes of all the data that is available to be returned to the user. The bytes returned field contains the length in bytes of all the data that is actually returned to the user.

All available data is returned if enough space is provided in the receiver variable. If the size of the receiver variable is at least large enough to contain all of the data, the bytes returned field equals the bytes available field. If the receiver variable is not large enough to contain all of the data, the bytes available field contains the number of bytes that can be returned.

Your code can check the values for both the bytes available and bytes returned fields. If the value of the bytes available field is greater than the value of the bytes returned field, the API has more information to return than what can fit in the receiver variable. This might occur, over time, because the APIs that you use might be enhanced with new releases. The API might also have more information to return if the receiver variable is being used to return a variable-length field (or array) and a very large value is returned on this API call. If both values are the same, the API returns all the information.

Depending on the capabilities of your high-level language, some API users take advantage of the following technique to avoid guessing the appropriate size for the receiver variable:

1. Call the API with a receiver variable length of 8 bytes (that is, just enough for the bytes available and the bytes returned fields).
2. Dynamically allocate an amount of storage equivalent to the bytes available.
3. Set the length of the receiver variable parameter to the amount of storage allocated.
4. Call the API a second time with the re-allocated, larger receiver variable.

This technique provides for highly flexible use of APIs that can return variable amounts of data.

Keyed interface:

Some APIs have a keyed interface for selecting what information you want returned. A keyed interface allows you to provide information to an API through the use of keys.

Keys are API-specific values that inform an API that a certain function should be performed. Keys are also used to pass information to an API or to retrieve information from an API.

Through the use of keys, you can be more selective; you can choose one item or a number of items rather than all of them. For example, using the List Job (QUSLJOB) API, you can receive selected information about a job based on the keys that you specify. If you want job information about the output queue priority, you only need to specify the output queue priority key.

Although there are some exceptions, the keys are typically supplied and passed to an API through a variable-length record. A *variable-length record* is a collection of information that specifies the key being used and the data that is associated with the key. If a given structure contains binary values, it must be 4-byte aligned.

Some APIs that use variable-length records in addition to the QUSLJOB API are the Change Object Description (QLICOBJD) API and the Register Exit Point (QUSRGPT, QusRegisterExitPoint) API. You can use the appropriate include file in member QUS in the system include (QSYSINC) library when you have variable-length records as either input or output.

A keyed interface provides an easy-to-use means for enhancing an API without affecting the user who chooses not to use the enhancements.

Related reference:

Example: Keyed interface using ILE APIs

"Example in ILE C: Using keys with the List Spooled Files (QUSLSPL) API" on page 177
This ILE C program processes a list of spooled file information that you have specified using keys.

"Example in ILE COBOL: Using keys with the List Spooled Files (QUSLSPL) API" on page 174
This ILE COBOL program processes a list of spooled file information that you have specified using keys.

"Example in ILE RPG: Using keys with the List Spooled Files (QUSLSPL) API" on page 181
This ILE RPG program processes a list of spooled file information that you have specified using keys.

"Example in OPM RPG: Using keys with the List Spooled Files (QUSLSPL) API" on page 168
This OPM RPG program processes a list of spooled file information that you have specified using keys.

User space alternative:

If the amount of information to be returned by a retrieve API is not known or is large, a user space is preferred to contain the information.

The disadvantage of using a receiver variable when it is too small for the amount of data being returned is that the API must be called again to receive the remaining data. You can create a user space so that it can automatically extend up to 16 MB of storage to accommodate the information being retrieved.

Continuation handle

When a call to an API is made and the API has more information to return than what can fit in the receiver variable or the user space, the API returns a continuation handle, which is used to mark the last value put in the receiver variable or the user space.

If a continuation handle is returned to the caller because there is more information to return, the caller can call the API again and pass the continuation handle that was returned. The API continues to return information from the point it left off on the call that generated the continuation handle.

When you use the continuation handle parameter, that is the only parameter that can change. All other parameters must appear as they did on the call to the API that generated the continuation handle to obtain predictable results.

To use a continuation handle, follow these steps:

1. Blank out the continuation handle to let the API know that this is a first attempt at the retrieve operation.
2. Call the API to retrieve the information.
3. Use the information returned.
4. If the continuation handle field in the receiver variable is not set to blanks, do the following steps until the continuation handle equals blanks:
   a. Copy the continuation handle from the receiver variable to the continuation handle parameter.
   b. Call the API again by using the continuation handle that was returned. Keep all other parameters the same as in the original API call.

For a program example that uses a continuation handle, see "Example in ILE C: Retrieving exit point and exit program information" on page 211.

Related reference:

"General data structure for list APIs"
The data structure for the list APIs consists of several fields.

List APIs overview

List APIs return a list of information. They use a common generic header to provide information such as the number of list entries and the size of a list entry. The content of the list is unique to each API.

Related reference:

List Objects That Adopt Owner Authority (QSYLOBJP) API

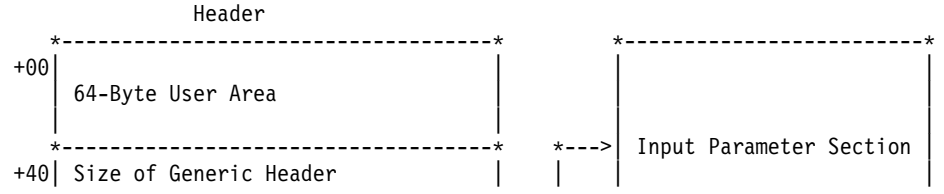
"User spaces" on page 67
List APIs return information to user spaces. A *user space* is an object consisting of a collection of bytes that can be used for storing any user-defined information.

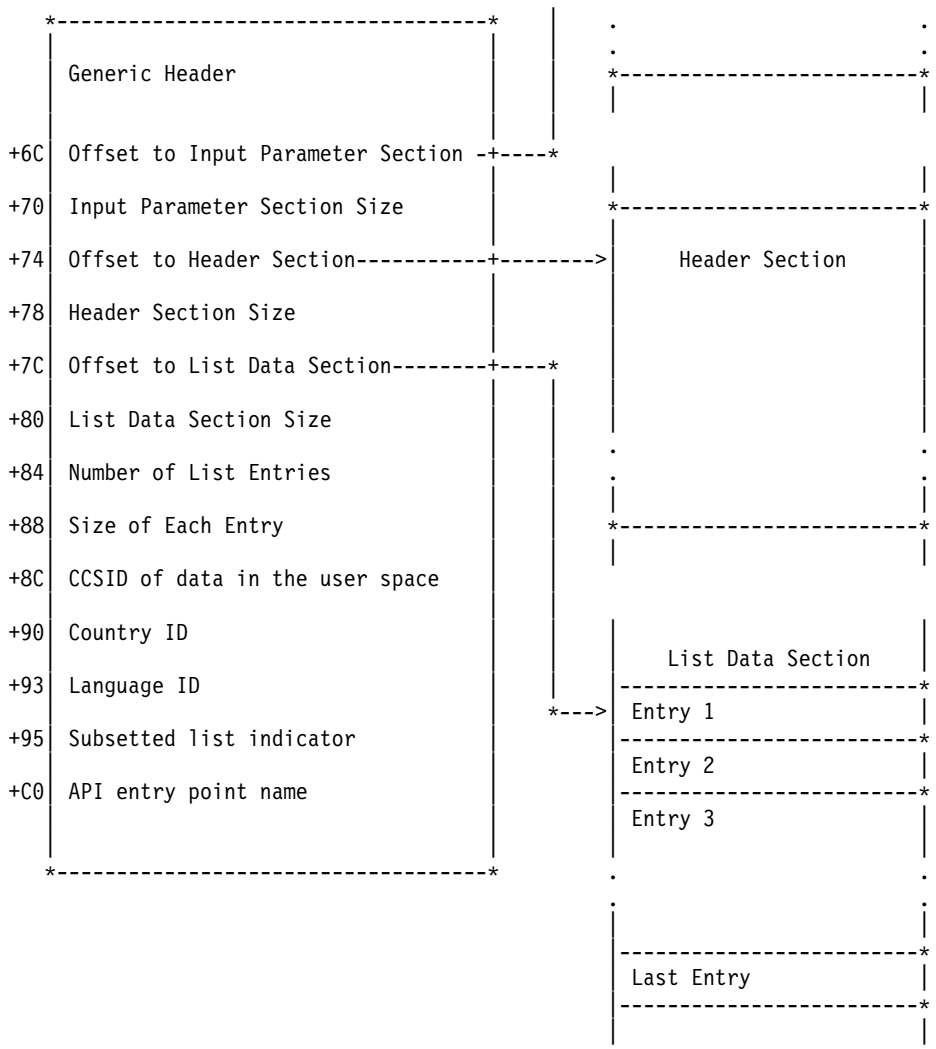
"Example in OPM RPG: Using keys with the List Spooled Files (QUSLSPL) API" on page 168
This OPM RPG program processes a list of spooled file information that you have specified using keys.

General data structure for list APIs

The data structure for the list APIs consists of several fields.

The following data structure shows the common fields that the list APIs use. All list APIs have an input parameter section, a header section, and a list data section.





User area

The first field in the general data structure is called the user area. This is a 64-byte field that is not used or changed by the system. Whatever information you place in this field remains there. For example, you can specify the date last used or include comments about the list.

Size of generic header

The size of the generic header does not include the size of the user area. All sections have a size, which might differ for each API.

Some fields might be added to the generic header from release to release. Because fields might be added, you might want to check the size of this field. If your application works across multiple releases, it is recommended that you check the size of this field to determine which fields are applicable.

Offset to input parameter section

The offset to the input parameter section is an offset to the start of the input parameter section. The input parameter section might contain a copy of the input parameters that you pass to a list API. For an example, see Input Parameter Section in List Objects That Adopt Owner Authority (QSYLOBJP) API.

The input parameter section contains a copy of the continuation handle value that you passed as the continuation handle parameter to the API. “Other fields of generic header” discusses continuation handles further.

Offset to header section

The header section includes an offset to where the header section starts and the size of the header section. This section is needed in the event any input parameters have a special value. The fields in the header section tell what the special value resolved to. For example, the special value *CURRENT for the user name parameter would resolve to the user profile name for the job that called the API.

This section is also sometimes used for API-specific control information that is not related to a particular list entry.

For an example, see Header Section in List Objects That Adopt Owner Authority (QSYLOBJP) API.

Offset to list data section

The offset to the list data section is the offset to the start of the format. The specific format that the API uses is determined by the name you specify for the format name parameter. The specific format that you use determines what information is returned in the user space.

The number of list entries field tells how many entries have been returned to you.

The size of each entry field within the list data section tells how large each entry is. In the list data section, each entry is of the same length for a given list. If the size of each entry field is `0`, the entries have different lengths and the format tells the length of each entry.

The list data sections for the QSYLOBJP API are shown in the OBJP0100 Format, OBJP0110 Format, and the OBJP0200 Format. This API has three possible formats.

For more information about formats and how to extract a field from a format, see Format and Extracting a field from the format.

Other fields of generic header

The field called *structure's release and level* is part of the generic header. This field describes the layout of the generic header. For a program-based API, this value should be 0100. For a service-program-based API, the value should be 0300.

The information status field tells you whether the information in the user space is complete and accurate, or partial. You need to check the value of this field before you do anything with the information in the user space, shown at (1) in the RPG example program. Possible values for this field follow.

Status value	Description
C	Complete and accurate.
I	Incomplete. The information you received is not accurate or complete.
P	Partial but accurate. The information you received is accurate, but the API had more information to return than the user space could hold.

If the value is `P`, the API has more information to return than what could fit in the user space. If you received the value `P`, you need to process the current information in the user space before you get the remaining information. The API returns a continuation handle usually in the form of a parameter. You

can use this continuation handle value to have the remaining information placed in the user space. You specify the continuation handle value that the API returned as the value of the continuation handle input parameter on your next call to the API.

The QSYLOBJP API provides a continuation handle in the header section to return the remaining information to the user space, as shown at (2) in the RPG example program. The user then passes this value back to the API as an input parameter so that the API can locate the remaining information and place it in the user space, as shown at (3) in the RPG example program.

If the API does not have a continuation handle and the information status field value is `P`, you must further qualify what you want in the list. In other words, you must be more specific on the parameter values that you pass to the API. For example, the QUSLOBJ API asked to get a list of objects; however, all of the objects on the system would not fit in the user space. To further qualify or limit the number of objects returned, the user might specify all libraries that start with a specific letter.

Related concepts:

"API format" on page 50
The format section in the API information shows the type of information to be returned by an API.

"Extracting a field from the format" on page 51
You can determine from the API format section where the field that you want to extract is located within the receiver variable.

Related tasks:

"Continuation handle" on page 76
When a call to an API is made and the API has more information to return than what can fit in the receiver variable or the user space, the API returns a continuation handle, which is used to mark the last value put in the receiver variable or the user space.

Related reference:

"Example in OPM RPG: List APIs" on page 87
This OPM RPG program prints a report that shows all objects that adopt owner authority.

User spaces for list APIs

List APIs require a user space to contain returned information.

A user space is an object type that is created by the Create User Space (QUSCRTUS) API. Generally, a user space is used when information about more than one object is being requested.

Most lists returned by APIs are made up of a series of entries where each entry is a data structure. Special fields are placed in the user space at consistent locations that describe:

- Where the list begins.
- The number of entries. Logic flow of processing a list of entries shows the logic for processing a list of entries.
- The length of each entry.

User spaces are used for such functions as returning either a list of members in a file or objects in a library. When you use one of the list APIs, the parameter list requires that you name the user space that will be used.

User spaces can be processed in two ways:

- If your language supports pointers, you can access or change the information directly. Language selection considerations describes each supported language and whether it supports pointers. Generally, pointer access is faster than API access.

- For languages that do not support pointers, you can use APIs to access or change the data in a user space. For example, the data in a user space can be accessed by the Retrieve User Space (QUSRTVUS) API. The API identifies a receiver variable that receives a number of bytes of information from the user space.

You can pass the user space as a parameter to a program. You do need to use a language that has pointer support to be able to pass the address of the first byte of the user space as a parameter to the processing program.

Related reference:

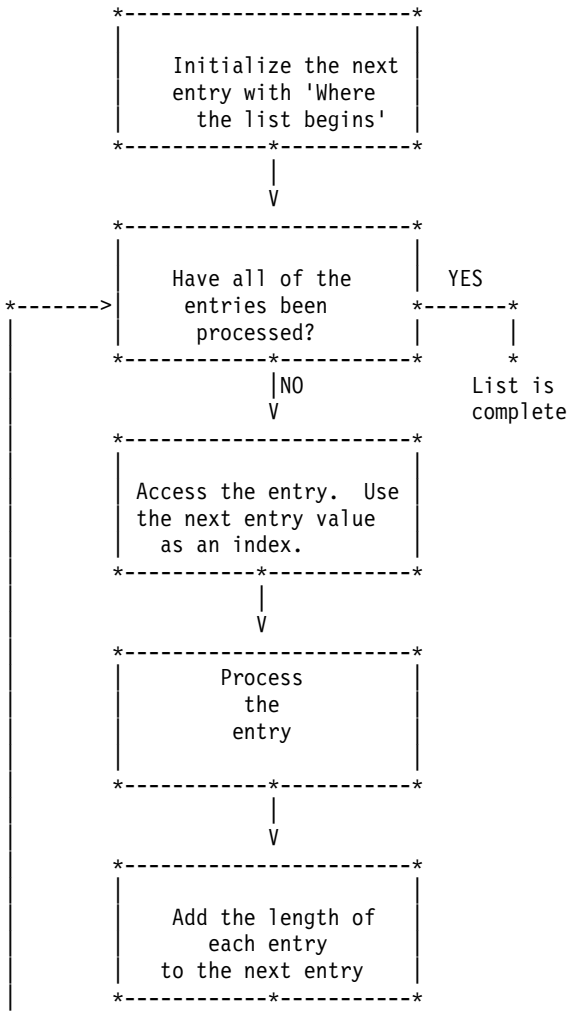
"User spaces" on page 67
List APIs return information to user spaces. A *user space* is an object consisting of a collection of bytes that can be used for storing any user-defined information.

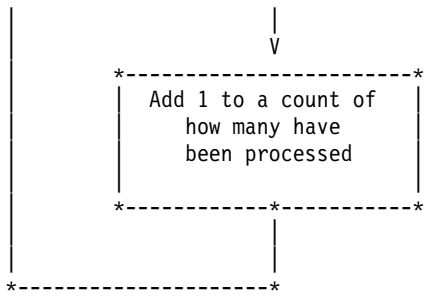
"Language selection considerations" on page 8
You can directly use APIs, other than service-program-based APIs, with all the languages that are available with the IBM i operating system.

Logic flow of processing a list of entries:

When you process a list of entries returned by a list API, do not statically encode the values. Use the offset, the length of each entry, and the number of entries so that applications are compatible with future releases.

This is the logic flow for processing a list that contains multiple entries:





Manipulating a user space with pointers:

Some high-level languages support pointers. A *pointer* is a data element or variable that holds the address of a data object or a function. Using pointers, you can manipulate information more rapidly from the user space.

The high-level languages that support pointers include ILE C, Visual Age for C++, ILE COBOL, ILE RPG, ILE CL, COBOL, CL, Pascal, and PL/I.

Synchronizing between two or more jobs

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

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

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

The preceding list is ordered by relative performance where CMPSWP is the fastest. If you do not synchronize two or more jobs, multiple concurrent updates to the user space or read operations might occur while information is being updated. As a result, the data might not be accurate.

Using offset values with pointers

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

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

See Example: Changing a user space with an ILE RPG program for an example of this procedure.

Updating usage data

If you are using the Change User Space (QUSCHGUS) or Retrieve User Space (QUSRTVUS) API to manipulate user spaces, you do not need to update usage data information. If you directly retrieve data using pointers, your application programs should update the usage data information. To do this, use the

QUSCHGUS API to update the date last changed and use the QUSRTVUS API to update the date last retrieved. You do not need to do this for each retrieve or change operation to the user space, but you should do this once within each application program to maintain accurate usage data information.

Related reference:

"Examples: Changing a user space"
These high-level language programs update the contents of a user space with and without using pointers.

Manipulating a user space without pointers:

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

Position values

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

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

Lengths

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

Using offset values with the change and retrieve user space APIs

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

Related reference:

"Example in OPM RPG: List APIs" on page 87
This OPM RPG program prints a report that shows all objects that adopt owner authority.

Examples: Changing a user space:

These high-level language programs update the contents of a user space with and without using pointers.

Related concepts:

"Manipulating a user space with pointers" on page 81
Some high-level languages support pointers. A *pointer* is a data element or variable that holds the address of a data object or a function. Using pointers, you can manipulate information more rapidly from the user space.

Example: User space before and after change:

This example compares the user space before and after you change it using the Change User Space (QUSCHGUS) API.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Here is the user space before you change it with one of the change examples:

```
xxxxSS1 VxRxMx yymmdd          System i5 Dump               128747/ERICJ/ERICJS1          03/22/07 11:03:26
DMPSYSOBJ PARAMETERS
   TEMPSPACE             CONTEXT-QGPL
OBJ-
                         *USRSPC
OBJTYPE-
OBJECT TYPE-                  SPACE                                  *USRSPC
NAME-           TEMPSPACE                           TYPE-             19   SUBTYPE-        34
LIBRARY-        QGPL                                TYPE-             04   SUBTYPE-        01
CREATION-       3/22/07  11:02:56                  SIZE-             0000400
OWNER-          ERICJ                               TYPE-             08   SUBTYPE-        01
ATTRIBUTES-             0800                        ADDRESS-          01841400  0000
SPACE ATTRIBUTES-
 000000  00000080  00000060  1934E3C5  D4D7E2D7  C1C3C540  40404040 40404040  40404040 *  -TEMPSPACE     *
 000020  40404040  40404040  E0000000  00000000  00000200  5C800000 00000000  00000000 *   \       *     *
 000040  00000000  00000000  00020002  6E000400  00000000  00000000 00000000  00000000 *     >           *
SPACE-
 0000E00 5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C 5C5C5C5C  5C5C5C5C ******************
        LINES  000200     TO     0001FF   SAME AS ABOVE

.POINTERS-
    NONE
OIR DATA-
.TEXT
 000000  E4A28599  40A29781  83854086  969940C3  88819587  8540E4A2 859940E2  97818385 *user space for Change User *
 000020  40C5A781  94979385                                                        *Space Example              *
.SERVICE-
 000000  40404040  40404040  40404040  40404040  40404040  40404040 40404040  40404040 *               1          *
 000020  40404040  40404040  404040D9  F0F3D4F0  F0F0F9F0  F0F3F2F2 F1F1F0F2  F5F64040 *       VxRxMx007032210256  *
 000040  40404040  40404040  40404040  40404040  40404040  40F14040 40404040  40404040 *
 000060  40404040  40404040  40404040  40404040  40404040  40404040 00000000  00000000 *
 000080  00000000  00000000  00000000  00000000  00000000  00000000 00000000  00000000 *
 0000A0  00000000  00000000
 END OF DUMP

                          * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Here is the user space after you change it with one of the change examples. The change takes place in SPACE-.

```
xxxxSS1 VxRxMx yymmdd          System i5 Dump               128747/ERICJ/ERICJS1          03/22/07 11:03:26
DMPSYSOBJ PARAMETERS
   TEMPSPACE             CONTEXT-QGPL
OBJ-
                         *USRSPC
OBJTYPE-
OBJECT TYPE-                  SPACE                                  *USRSPC
NAME-           TEMPSPACE                           TYPE-             19   SUBTYPE-        34
LIBRARY-        QGPL                                TYPE-             04   SUBTYPE-        01
CREATION-       3/22/07  11:02:56                  SIZE-             0000400
OWNER-          ERICJ                               TYPE-             08   SUBTYPE-        01
ATTRIBUTES-             0800                        ADDRESS-          01841400  0000
SPACE ATTRIBUTES-
 000000 00000080  00000060  1934E3C5  D4D7E2D7  C1C3C540  40404040 40404040  40404040 *  -TEMPSPACE
 000020 40404040  40404040  E0000000  00000000  00000200  5C800000 00000000  00000000 *   \       *                *
 000040 00000000  00000000  00020002  6E000400  00000000  00000000 00000000  00000000 *     >                        *
SPACE-
 000000 C2898740  E2A39989  95874097  81848485  8440A689  A3884082 93819592  A2404040 *Big string padded with blanks*
 000020 40404040  40404040  40404040  40404040  40404040  40404040 40404040  40404040
 000040 5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C  5C5C5C5C 5C5C5C5C  5C5C5C5C ********************************
        LINES  000060     TO     0001FF   SAME AS ABOVE

.POINTERS-
    NONE
OIR DATA-
.TEXT
 000000  E4A28599  40A29781  83854086  969940C3  88819587  8540E4A2 859940E2  97818385 *user space for Change User *
 000020  40C5A781  94979385                                                        *Space Example              *
.SERVICE-
 000000  40404040  40404040  40404040  40404040  40404040  40404040 40404040  40404040 *               1          *
 000020  40404040  40404040  404040D9  F0F3D4F0  F0F0F9F0  F0F3F2F2 F1F1F0F2  F5F64040 *       VxRxMx007032210256  *
 000040  40404040  40404040  40404040  40404040  40404040  40F14040 40404040  40404040 *
 000060  40404040  40404040  40404040  40404040  40404040  40404040 00000000  00000000 *
 000080  00000000  00000000  00000000  00000000  00000000  00000000 00000000  00000000 *
 0000A0  00000000  00000000
```

```
END OF DUMP

                    * * * * *  E N D  O F  L I S T I N G  * * * * *
```

Example in ILE RPG: Changing a user space:

This ILE RPG program changes the user area of a user space using a pointer.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
H*****************************************************************
H*
H* PROGRAM: CHANGUSPTR
H*
H* LANGUAGE: ILE RPG for IBM i
H*
H* DESCRIPTION: CHANGE THE CONTENTS OF INFORMATION IN THE USER
H*              AREA IN THE USER SPACE USING A POINTER
H*
H*****************************************************************
D*
DUSRSPCNAM        S             20    INZ('TEMPSPACE QTEMP     ')
DNEWVALUE         S             64    INZ('Big String padded with blanks')
DUSRSPCPTR        S               *
DUSERAREA         DS                  BASED(USRSPCPTR)
D CHARFIELD               1     64
D*
D* Following QUSEC structure copied from QSYSINC library
D*
DQUSEC            DS
D*                                            Qus EC
D QUSBPRV                 1      4B 0
D*                                            Bytes Provided
D QUSBAVL                 5      8B 0
D*                                            Bytes Available
D QUSEI                   9     15
D*                                            Exception Id
D QUSERVED               16     16
D*                                            Reserved
D* End of QSYSINC copy
D*
C*
C* Initialize Error code structure to return error ids
C*
C                   Z-ADD     16            QUSBPRV
C*
C* Set USRSPCPTR to the address of the user space
C*
C                   CALL      'QUSPTRUS'
C                   PARM                    USRSPCNAM
C                   PARM                    USRSPCPTR
C                   PARM                    QUSEC
C*
C* Check for successful setting of pointer
C*
C     QUSBAVL       IFGT      0
C*
C* If an error, then display the error message id
C*
C                   DSPLY                   QUSEI
C                   ELSE
C*
C* Otherwise, update the user space via the based structure
C*
C                   MOVEL     NEWVALUE      USERAREA
C                   END
```

```
C*
C* And return to our caller
C*
C                   SETON                                        LR
C                   RETURN
```

Example in OPM RPG: Changing a user space:

This OPM RPG program changes the user area of a user space without using a pointer.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
H*****************************************************************
H*****************************************************************
H*                                                               *
H*  PROGRAM:  CHANGUS                                            *
H*                                                               *
H*  LANGUAGE:  RPG                                               *
H*                                                               *
H*  DESCRIPTION:  THIS PROGRAM WILL CHANGE THE CONTENTS OF       *
H*                INFORMATION IN THE USER AREA IN THE USER SPACE*
H*                (FIRST 64 BYTES).                              *
H*                                                               *
H*  APIs USED:  QUSCHGUS                                         *
H*                                                               *
H*****************************************************************
H*****************************************************************
E                    ARY     1   1 20
E                    CHG     1   1 64
IUSRSPC      DS
I                                      1  10 USNAME
I                                     11  20 USLIB
I            DS
I                                   B   1   40LENDTA
I                                   B   5   80STRPOS
C*                                                               *
C*****************************************************************
C*****************************************************************
C*                                                               *
C*              OPERABLE CODE STARTS HERE                        *
C*                                                               *
C*****************************************************************
C*****************************************************************
C*                                                               *
C* MOVE THE USER SPACE AND LIBRARY NAME FROM ARY ARRAY INTO THE  *
C* USRSPC DATA STRUCTURE.  ALSO, MOVE THE NEW USER DATA FROM     *
C* CHG ARRAY INTO NEWVAL.                                        *
C*                                                               *
C                    MOVELARY,1     USRSPC
C                    MOVELCHG,1     NEWVAL  64
C*                                                               *
C                    Z-ADD64        LENDTA           LEN OF USERAREA
C                    Z-ADD1         STRPOS           STARTING POS
C                    MOVE '1'       FORCE   1        FORCE PARM
C*                                                               *
C* CALL THE QUSCHGUS API WHICH WILL CHANGE THE USER AREA IN THE  *
C* USER SPACE.                                                   *
C*                                                               *
C                    CALL 'QUSCHGUS'
C                    PARM           USRSPC
C                    PARM           STRPOS
C                    PARM           LENDTA
C                    PARM           NEWVAL
C                    PARM           FORCE
C*                                                               *
```

```
     C* IF MORE OF THE USER SPACE NEEDS TO BE CHANGED, THIS PROGRAM  *
     C* COULD BE UPDATED TO LOOP UNTIL THE END OF THE ARRAY WAS      *
     C* REACHED.                                                     *
     C*                                                             *
     C                     SETON                     LR
     C                     RETRN
** ARY
TEMPSPACE QGPL
** CHG
Big String padded with blanks
```

Additional information about list APIs and user spaces:

A list API returns only the number of list entries that can fit inside the user space. If you have *CHANGE authority to the user space, the list API can extend the user space when it is too small to contain the list.

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

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

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

Example in CL: Listing database file members:

This CL program generates a list of members in a database file that start with M and places the list in a user space.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

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

```
 /*   QCLSRC FROM LIBRARY QGPL USING THE OUTPUT FORMAT MBRL0200. */
 /*   OVERRIDE PROCESSING SHOULD OCCUR.                          */
 /*************************************************************/
      CALL QUSLMBR                                                +
           ('EXAMPLE   QGPL      ' /* USER SPACE NAME AND LIB    */ +
            'MBRL0200'             /* FORMAT NAME                */ +
            'QCLSRC    QGPL      ' /* DATABASE FILE AND LIBRARY  */ +
            'M*        '           /* MEMBER NAME                */ +
            '1')                   /* OVERRIDE PROCESSING        */
ENDPGM
```

Example in OPM RPG: List APIs

This OPM RPG program prints a report that shows all objects that adopt owner authority.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*
F*****************************************************************
F*****************************************************************
F*****************************************************************
F*****************************************************************
F*
F*Program Name: List objects which adopt owner authority
F*
F*Language:     OPM RPG
F*
F*Description:  This program prints a report showing all objects
F*              that adopt owner authority.  The two parameters
F*              passed to the program are the profile to be
F*              checked and the type of objects to be listed.
F*              The parameter values are the same as those
F*              accepted by the QSYLOBJP API.
F*
F*APIs Used:    QSYLOBJP - List Objects that Adopt Owner Authority
F*              QUSCRTUS - Create User Space
F*              QUSROBJD - Retrieve Object Description            /
F*              QUSRTVUS - Retrieve From User Space               /
F*
F*****************************************************************
F*****************************************************************
F*
FQSYSPRT O   F     132     OF     PRINTER
F*************************************************************
I/COPY QSYSINC/QRPGSRC,QSYLOBJP
I/COPY QSYSINC/QRPGSRC,QUSROBJD
I/COPY QSYSINC/QRPGSRC,QUSGEN
C*****************************************************************
I* Error Code Structure
I*
I* This shows how the user can define the variable length portion
I* of error code for the exception data.
I*
I*/COPY QSYSINC/QRPGSRC,QUSEC
I*** START HEADER FILE SPECIFICATIONS ****************************
I*
I*Header File Name: H/QUSEC
I*
I*Descriptive Name: Error Code Parameter.
I*
I*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
I*All rights reserved.
I*US Government Users Restricted Rights -
I*Use, duplication or disclosure restricted
I*by GSA ADP Schedule Contract with IBM Corp.
I*
I*Licensed Materials-Property of IBM
```

```
I*
I*
I*Description: Include header file for the error code parameter.
I*
I*Header Files Included: None.
I*
I*Macros List: None.
I*
I*Structure List: Qus_EC_t
I*                Qus_ERRC0200_t
I*
I*Function Prototype List: None.
I*
I*Change Activity:
I*
I*CFD List:
I*
I*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
I*---- ------------ ----- ------ --------- ----------------------
I*$A0= D2862000     3D10  931201 DPOHLSON: New Include
I*$B1= D9179400     3D60  940904 GEORGE  : Add Qus_ERRC0200_t
I*                                        structure.
I*
I*End CFD List.
I*
I*Additional notes about the Change Activity
I*End Change Activity.
I*** END HEADER FILE SPECIFICATIONS *****************************
I****************************************************************
I*Record structure for Error Code Parameter
I****                                                          ***
I*NOTE: The following type definition only defines the fixed
I*   portion of the format.  Varying length field Exception
I*   Data will not be defined here.
I****************************************************************
IQUSBN       DS
I*                                             Qus EC
I                                     B   1   40QUSBNB
I*                                             Bytes Provided
I                                     B   5   80QUSBNC
I*                                             Bytes Available
I                                         9  15 QUSBND
I*                                             Exception Id
I                                        16  16 QUSBNF
I*                                             Reserved
I*  Following statement was uncommented and 17 was changed to 100
I                                        17 100 QUSBNG
I*
I*                                       Varying length
IQUSKY       DS
I*                                             Qus ERRC0200
I                                     B   1   40QUSKYB
I*                                             Key
I                                     B   5   80QUSKYC
I*                                             Bytes Provided
I                                     B   9  120QUSKYD
I*                                             Bytes Available
I                                        13  19 QUSKYF
I*                                             Exception Id
I                                        20  20 QUSKYG
I*                                             Reserved
I                                     B  21  240QUSKYH
I*                                             CCSID
I                                     B  25  280QUSKYJ
I*                                             Offset Exc Data
I                                     B  29  320QUSKYK
I*                                             Length Exc Data
```

```
I*                                     33  33 QUSKYL
I*                                        Reserved2
I*
I*                                     34  34 QUSKYM
I*
I*
I* Global Variables
I*
I            DS
I                                       1  10 APINAM
I                                      11  30 CONHDL
I I            'QSYSLOBJP '            31  40 EXTATR
I                                      41  41 LSTSTS
I I            'OBJP0200'              42  49 MBRLST
I I            'OBJD0100'              68  75 RJOBDF
I I            '*ALL      '            76  85 SPCAUT
I I            '*USER     '            86  95 SPCDMN
I I            X'00'                   96  96 SPCINT
I I            'ADOPTS    QTEMP     '  97 116 SPCNAM
I I            '*YES      '           117 126 SPCREP
I                                     127 176 SPCTXT
I I            '*USRSPC   '           177 186 SPCTYP
I I            8                     B 197 2000RCVLEN
I                                    B 201 2040SIZENT
I I            1                     B 205 2080SPCSIZ
I                                    B 209 2120I
I                                    B 213 2160NUMENT
I                                    B 217 2200OFFSET
I                                    B 221 2240STRPOS
IRCVVAR      DS                          2000
C*
C* Beginning of Mainline
C*
C* Two parameters are being passed into this program.
C*
C            *ENTRY    PLIST
C                      PARM           USRPRF 10
C                      PARM           OBJTYP 10
C*
C*****************************************************************
C                      EXSR INIT
C                      EXSR PROCES
C                      EXSR DONE
C*
C* End of MAINLINE
C*
C*
C*****************************************************************
C* Function:      getlst
C*
C* Description:   This function calls QSYLOBJP to build a list.
C*
C*****************************************************************
C*
C            GETLST    BEGSR
C                      MOVEL'OBJP0200'MBRLST
C*****************************************************************
C* Call QSYLOBJP API to generate a list.  The continuation handle
C* is set by the caller of this function.
C*****************************************************************
C                      CALL 'QSYLOBJP'
C                      PARM           SPCNAM          User space/lib
C                      PARM           MBRLST          Member list
C                      PARM           USRPRF          User profile
C                      PARM           OBJTYP          Object type sc
C                      PARM           CONHDL          Continuation ha    (3)
C                      PARM           QUSBN           Error Code
```

```
C*****************************************************************
C* Check for errors on QSYLOBJP.
C*****************************************************************
C           QUSBNC    IFGT 0
C                     MOVEL'QSYLOBJP'APINAM
C                     EXSR APIERR
C                     ENDIF
C                     ENDSR
C*****************************************************************
C* Function:      INIT
C*
C* Description:   This function does all the necessary
C*                initialization for this program and the
C*                rest is done in the I specs.
C*****************************************************************
C           INIT      BEGSR
C*****************************************************************
C                     Z-ADD100       QUSBNB
C*****************************************************************
C* Call QUSROBJD to see if the user space was previously created
C* in QTEMP.  If it was, simply reuse it.
C*****************************************************************
C                     CALL 'QUSROBJD'
C                     PARM           RCVVAR          Receiver Var
C                     PARM           RCVLEN          Rec Var Length
C                     PARM           RJOBDF          Format
C                     PARM           SPCNAM          Qual User Space
C                     PARM           SPCTYP          User object typ
C                     PARM           QUSBN           Error Code
C*
C           QUSBNC    IFGT 0
C*****************************************************************
C* If a CPF9801 error was received, then the user space was not
C* found.
C*****************************************************************
C           QUSBND    IFEQ 'CPF9801'
C*****************************************************************
C*  Create a user space for the list generated by QSYLOBJP.
C*****************************************************************
C                     CALL 'QUSCRTUS'
C                     PARM           SPCNAM          Qual User Space
C                     PARM           EXTATR          Extended Attrib
C                     PARM           SPCSIZ          Size user space
C                     PARM           SPCINT          Space Initializ
C                     PARM           SPCAUT          Public Authorit
C                     PARM           SPCTXT          User space text
C                     PARM           SPCREP          Replace existin
C                     PARM           QUSBN           Error Code
C                     PARM           SPCDMN          Domain of us
C*****************************************************************
C* Check for errors on QUSCRTUS.
C*****************************************************************
C           QUSBNC    IFGT 0
C                     MOVEL'QUSCRTUS'APINAM
C                     EXSR APIERR
C                     ENDIF
C*****************************************************************
C* An error occurred accessing the user space.
C*****************************************************************
C                     ELSE
C                     MOVEL'QUSROBJD'APINAM
C                     EXSR APIERR
C                     ENDIF                          CPF9801 ELSE
C                     ENDIF                          BYTAVL > 0
C*****************************************************************
C* Set QSYLOBJP (via GETLST) to start a new list.
C*****************************************************************
```

```
C                     MOVE *BLANKS   CONHDL
C                     EXSR GETLST
C*****************************************************************
C* Let's retrieve the generic header information from the user
C* space since OPM RPG does not have pointer support.
C*****************************************************************
C                     Z-ADD1         STRPOS
C                     Z-ADD192       RCVLEN           Format 100
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM           Qual User Space
C                     PARM           STRPOS           Start Position
C                     PARM           RCVLEN           Length of Data
C                     PARM           QUSBP            Receiver Var.
C                     PARM           QUSBN            Error Code
C*****************************************************************
C* Check for errors on QUSRTVUS.
C*****************************************************************
C           QUSBNC    IFGT 0
C                     MOVEL'QUSRTVUS'APINAM
C                     EXSR APIERR
C                     ENDIF
C           1         ADD  QUSBPQ    STRPOS           Offset to List      (5)
C                     ENDSR
C*****************************************************************
C* Function:      proc2
C*
C* Description:   This function processes each entry returned by
C*                QSYLOBJP.
C*
C*****************************************************************
C           PROC2     BEGSR
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM           Qual User Space
C                     PARM           STRPOS           Start Position
C                     PARM           SIZENT           Length of Data
C                     PARM           QSYB6            Receiver Var.
C                     PARM           QUSBN            Error Code
C*****************************************************************
C* Check for errors on QUSRTVUS.
C*****************************************************************
C           QUSBNC    IFGT 0
C                     MOVEL'QUSRTVUS'APINAM
C                     EXSR APIERR
C                     ENDIF
C                     EXCPTPRTENT
C*****************************************************************
C* After each entry, increment to the next entry.
C*****************************************************************
C           STRPOS    ADD  SIZENT    STRPOS                        (7)
C                     ENDSR
C*****************************************************************
C* Function:      proc1
C*
C* Description:   This function processes each entry returned by
C*                QSYLOBJP.
C*
C*****************************************************************
C           PROC1     BEGSR
C*****************************************************************
C* If valid information was returned.                             (1)
C*****************************************************************
C                     Z-ADDQUSBPS    NUMENT
C           QUSBPJ    IFEQ 'P'
C           QUSBPJ    OREQ 'C'
C           NUMENT    IFGT 0
C*****************************************************************
C* Get the size of each entry to use later.                        (4)
```

```
C*****************************************************************
C                     Z-ADDQUSBPT    SIZENT
C*****************************************************************
C* Increment to the first list entry.
C*****************************************************************
C           1         ADD  QUSBPQ    OFFSET
C*****************************************************************
C* Process all of the entries.
C*****************************************************************
C           1         DO   NUMENT    I                              (6)
C                     EXSR PROC2
C                     ENDDO
C*****************************************************************
C* If all entries in this user space have been processed, check
C* if more entries exist than can fit in one user space.
C*****************************************************************
C           QUSBPJ    IFEQ 'P'
C*****************************************************************
C* Address the input parameter header.
C*****************************************************************
C           1         ADD  QUSBPL    STRPOS
C                     Z-ADD68        RCVLEN           Format 100
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM           Qual User Space
C                     PARM           STRPOS           Start Position
C                     PARM           RCVLEN           Length of Data
C                     PARM           QUSBP            Receiver Var.
C                     PARM           QUSBN            Error Code
C*****************************************************************
C* Check for errors on QUSRTVUS.
C*****************************************************************
C           QUSBNC    IFGT 0
C                     MOVEL'QUSRTVUS'APINAM
C                     EXSR APIERR
C                     ENDIF
C*****************************************************************
C* If the continuation handle in the input parameter header
C* is blank, then set the list status to complete.
C*****************************************************************
C           QSYCRJ    IFEQ *BLANKS
C                     MOVE 'C'       LSTSTS
C                     ELSE
C*****************************************************************
C* Else, call QSYLOBJP reusing the user space to get more
C* list entries.
C*****************************************************************
C                     MOVELQSYCRJ    CONHDL                         (2)
C                     EXSR GETLST
C                     Z-ADD1         STRPOS
C                     Z-ADD192       RCVLEN           Format 100
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM           Qual User Space
C                     PARM           STRPOS           Start Position
C                     PARM           RCVLEN           Length of Data
C                     PARM           QUSBP            Receiver Var.
C                     PARM           QUSBN            Error Code
C*****************************************************************
C* Check for errors on QUSRTVUS.
C*****************************************************************
C           QUSBNC    IFGT 0
C                     MOVEL'QUSRTVUS'APINAM
C                     EXSR APIERR
C                     ENDIF
C                     MOVE QUSBPJ    LSTSTS
C                     ENDIF                           HDL = BLANKS
C                     ENDIF                           INFOSTS = 0
C                     ELSE
```

```
C*****************************************************************
C*If there exists an unexpected status, log an error (not shown)
C*and exit.
C*****************************************************************
C                     EXSR DONE                     done();
C                     ENDIF                         #ENT > 0
C                     ENDIF                         USRSPC=P/C
C                     ENDSR
C*****************************************************************
C* Function:      proces
C*
C* Description:   Processes entries until they are complete.
C*
C*****************************************************************
C           PROCES    BEGSR
C                     MOVELQUSBPJ    LSTSTS
C           LSTSTS    DOUEQ'C'
C           LSTSTS    OREQ 'I'
C                     EXSR PROC1                    proces1();
C                     ENDDO
C                     ENDSR
C*****************************************************************
C* Function:      done
C*
C* Description:   Exits the program.
C*
C*****************************************************************
C           DONE      BEGSR
C                     EXCPTENDLST
C                     SETON                    LR
C                     ENDSR
C*****************************************************************
C* Function:      apierr
C*
C* Description:   This function prints the API name, and exception
C*                identifier of an error that occurred.
C*****************************************************************
C           APIERR    BEGSR
C           APINAM    DSPLY
C           QUSBND    DSPLY
C                     EXSR DONE
C                     ENDSR
O*****************************************************************
O* Function:      PRTENT
O*
O* Description:   This function prints the information returned in
O*                user space.
O*****************************************************************
OQSYSPRT E  106           PRTENT
O                                       'Object: '
O                         QSYB6C
O                                       'Library: '
O                         QSYB6D
O                                       'Type: '
O                         QSYB6F
O                                       'Text: '
O                         QSYB6J
O*****************************************************************
O* Function:      ENDLST
O*
O* Description:   This function prints the end of listing print
O*                line and returns to the caller.
O*****************************************************************
OQSYSPRT E  106           ENDLST
O                                       '*** End of List'
```

The value in the information status field is shown at (1). The continuation handle in the header section to return the remaining information to the user space is shown at (2). The user then passes this value back to the API as an input parameter so that the API can locate the remaining information and place it in the user space, as shown at (3).

Processing a list

This is the preferred method for processing lists. To correctly process through a list, take the following actions:

1. Use the offset to list data section field (5).
2. Look at the number of list entries field in the list (6).
3. For processing lists with fixed-length entries, add the size of each entry field to get to the start of the next entry (7).
4. For variable-length entries, add the length of the entry (or displacement in some cases) to the next entry.

IBM might add fields to the bottom of formats in future releases. If this occurs and your code uses the size of each entry for a previous release, your list will not process at the start of each entry.

The example program defines the size of each entry at (4).

Related concepts:

"Manipulating a user space without pointers" on page 82
When programming in a high-level language that does not support pointers, you can use the Change User Space (QUSCHGUS) and Retrieve User Space (QUSRTVUS) APIs to manipulate data. However, you must first understand how to use positions and lengths with these APIs.

Related reference:

"General data structure for list APIs" on page 76
The data structure for the list APIs consists of several fields.

"Defining list-entry format lengths" on page 539
When you define the list-entry format length, the most common error is to statically encode the format length in your program. Here are the program examples that show the incorrect and correct ways of defining list-entry format lengths.

"Example in ILE COBOL: List APIs" on page 109
This ILE COBOL program prints a report that shows all objects that adopt owner authority.

"Example in ILE C: List APIs" on page 98
This ILE C program prints a report that shows all objects that adopt owner authority.

"Example in ILE RPG: List APIs" on page 104
This ILE RPG program prints a report that shows all objects that adopt owner authority.

Example in ILE CL: List APIs

This ILE CL program prints a report that shows all objects that adopt owner authority.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*****************************************************************/
/*                                                               */
/* Program:      List objects which adopt owner authority        */
/*                                                               */
/* Language:     ILE CL                                          */
/*                                                               */
/* Description:  This program displays all objects that adopt    */
/*               owner authority.  The two parameters passed to  */
/*               the program are the profile to be checked and   */
/*               the type of objects to be listed.  The parameter*/
/*               values are the same as those accepted by the    */
```

```
/*               QSYLOBJP API                                          */
/*                                                                     */
/*  APIs Used:   QSYLOBJP - List Objects that Adopt Owner Authority */
/*               QUSCRTUS - Create User Space                          */
/*               QUSPTRUS - Retrieve Pointer to User Space             */
/*               QUSROBJD - Retrieve Object Description                */
/*                                                                     */
/*********************************************************************/
             PGM        PARM(&USR_PRF &OBJ_TYPE)
             DCL        VAR(&USR_PRF) TYPE(*CHAR) LEN(10)
             DCL        VAR(&OBJ_TYPE) TYPE(*CHAR) LEN(10)
             DCL        VAR(&ERRCDE) TYPE(*CHAR) LEN(16)
             DCL        VAR(&BYTPRV) TYPE(*INT) STG(*DEFINED) LEN(4) +
                          DEFVAR(&ERRCDE)
             DCL        VAR(&BYTAVL) TYPE(*INT) STG(*DEFINED) LEN(4) +
                          DEFVAR(&ERRCDE 5)
             DCL        VAR(&MSGID) TYPE(*CHAR) STG(*DEFINED) LEN(7) +
                          DEFVAR(&ERRCDE 9)
             DCL        VAR(&RCVVAR) TYPE(*CHAR) LEN(8)
             DCL        VAR(&RCVVARSIZ) TYPE(*INT) LEN(4) VALUE(8)
             DCL        VAR(&SPC_NAME) TYPE(*CHAR) LEN(20) +
                          VALUE('ADOPTS    QTEMP     ')
             DCL        VAR(&SPC_SIZE) TYPE(*INT) LEN(4) VALUE(1)
             DCL        VAR(&SPC_INIT) TYPE(*CHAR) LEN(1) VALUE(X'00')
             DCL        VAR(&BLANKS) TYPE(*CHAR) LEN(50)
             DCL        VAR(&CONTIN_HDL) TYPE(*CHAR) LEN(20)
             DCL        VAR(&SPCPTR) TYPE(*PTR)
             DCL        VAR(&LISTHDR) TYPE(*CHAR) STG(*BASED) +
                          LEN(192) BASPTR(&SPCPTR)
             DCL        VAR(&LISTSTS) TYPE(*CHAR) STG(*DEFINED) +
                          LEN(1) DEFVAR(&LISTHDR 104)
             DCL        VAR(&PARMHDROFS) TYPE(*INT) STG(*DEFINED) +
                          LEN(4) DEFVAR(&LISTHDR 109)
             DCL        VAR(&LISTENOFS) TYPE(*INT) STG(*DEFINED) +
                          DEFVAR(&LISTHDR 125)
             DCL        VAR(&LISTENTNBR) TYPE(*INT) STG(*DEFINED) +
                          DEFVAR(&LISTHDR 133)
             DCL        VAR(&LISTENTSIZ) TYPE(*INT) STG(*DEFINED) +
                          DEFVAR(&LISTHDR 137)
             DCL        VAR(&LST_STATUS) TYPE(*CHAR) LEN(1)
             DCL        VAR(&LSTPTR) TYPE(*PTR)
             DCL        VAR(&LSTENT) TYPE(*CHAR) STG(*BASED) +
                          LEN(100) BASPTR(&LSTPTR)
             DCL        VAR(&OBJECT) TYPE(*CHAR) STG(*DEFINED) +
                          LEN(10) DEFVAR(&LSTENT 1)
             DCL        VAR(&CONTIN) TYPE(*CHAR) STG(*DEFINED) +
                          LEN(20) DEFVAR(&LSTENT 11)
             DCL        VAR(&CURENT) TYPE(*INT) LEN(4)
             CALLSUBR   SUBR(INIT)
             CALLSUBR   SUBR(PROCES)
             RETURN

             SUBR       SUBR(PROCES)
/*                                                                     */
/* This subroutine processes each entry returned by QSYLOBJP         */
/*                                                                     */
/* Do until the list is complete                                       */
/*                                                                     */
             CHGVAR     VAR(&LST_STATUS) VALUE(&LISTSTS)
             DOUNTIL    COND(&LST_STATUS *EQ 'C')
             IF         COND((&LISTSTS *EQ 'C') *OR (&LISTSTS *EQ +
                          'P')) THEN(DO)
/*                                                                     */
/* And list entries were found                                         */
/*                                                                     */
             IF         COND(&LISTENTNBR *GT 0) THEN(DO)
/*                                                                     */
```

```
/* Set &LSTPTR to first byte of the User Space                       */
/*                                                                  */
             CHGVAR     VAR(&LSTPTR) VALUE(&SPCPTR)
/*                                                                  */
/* Increment &LSTPTR to the first list entry                        */
/*                                                                  */
             CHGVAR     VAR(%OFFSET(&LSTPTR)) VALUE(%OFFSET(&LSTPTR) +
                          + &LISTENTOFS)
/*                                                                  */
/* And process all the entries                                      */
/*                                                                  */
             DOFOR      VAR(&CURENT) FROM(1) TO(&LISTENTNBR)
             SNDPGMMSG  MSG(&OBJECT) TOPGMQ(*EXT)
/*                                                                  */
/* After each entry, increment &LSTPTR to the next entry            */
/*                                                                  */
             CHGVAR     VAR(%OFFSET(&LSTPTR)) +
                          VALUE(%OFFSET(&LSTPTR) + &LISTENTSIZ)
             ENDDO
             ENDDO
/*                                                                  */
/* If all entries in this list have been processed, check if        */
/* more entries exist than can fit in one User Space                */
/*                                                                  */
             IF         COND(&LISTSTS *EQ 'P') THEN(DO)
/*                                                                  */
/* by reseting LSTPTR to the start of the User Space                */
/*                                                                  */
             CHGVAR     VAR(&LSTPTR) VALUE(&SPCPTR)
/*                                                                  */
/* and then incrementing &LSTPTR to Input Parameter Header          */
/*                                                                  */
             CHGVAR     VAR(%OFFSET(&LSTPTR)) VALUE(%OFFSET(&LSTPTR) +
                          + &PARMHDROFS)
/*                                                                  */
/* if the continuation handle is blank then the list is complete    */
/*                                                                  */
             IF         COND(&CONTIN *EQ ' ') THEN(CHGVAR +
                          VAR(&LST_STATUS) VALUE('C'))
             ELSE       CMD(DO)
/*                                                                  */
/* call QSYLOBP to get more entries                                 */
/*                                                                  */
             CHGVAR     VAR(&CONTIN_HDL) VALUE(&CONTIN)
             CALLSUBR   SUBR(GETLST)
             CHGVAR     VAR(&LST_STATUS) VALUE(&LISTSTS)
             ENDDO
             ENDDO
             ENDDO
             ELSE       CMD(DO)
/*                                                                  */
/* and if unexpected status, log an error                           */
/*                                                                  */
             SNDPGMMSG  MSG('Unexpected status') TOPGMQ(*EXT)
             RETURN
             ENDDO
             ENDDO
             ENDSUBR

             SUBR       SUBR(GETLST)
/*                                                                  */
/* Call QSYLOBJP to generte a list                                  */
/* The continuation handle is primed by the caller of this          */
/* subroutine                                                       */
/*                                                                  */
             CALL       PGM(QSYLOBJP) PARM(&SPC_NAME 'OBJP0200' +
                          &USR_PRF &OBJ_TYPE &CONTIN_HDL &ERRCDE)
```

```
/*                                                                   */
/* Check for errors on QSYLOBJP                                      */
/*                                                                   */
             IF         COND(&BYTAVL *GT 0) THEN(DO)
             SNDPGMMSG  MSG('Failure with QSYLOBJP') TOPGMQ(*EXT)
             RETURN
             ENDDO
             ENDSUBR

             SUBR       SUBR(INIT)
/*                                                                   */
/* One time initialization code for this program                     */
/*                                                                   */
/* Set Error Code structure not to use exceptions                    */
/*                                                                   */
             CHGVAR     VAR(&BYTPRV) VALUE(16)
/*                                                                   */
/* Check if the User Space was previously created                    */
/*                                                                   */
             CALL       PGM(QUSROBJD) PARM(&RCVVAR &RCVVARSIZ +
                          'OBJD0100' &SPC_NAME '*USRSPC' &ERRCDE)
/*                                                                   */
/* Check for errors on QUSROBJD                                      */
/*                                                                   */
             IF         COND(&BYTAVL *GT 0) THEN(DO)
/*                                                                   */
/* If CPF9801, then User Space not found                             */
/*                                                                   */
             IF         COND(&MSGID *EQ 'CPF9801') THEN(DO)
/*                                                                   */
/* So create a User Space for the list generated by QSYLOBJP         */
/*                                                                   */
             CALL       PGM(QUSCRTUS) PARM(&SPC_NAME 'QSYLOBJP' +
                          &SPC_SIZE &SPC_INIT '*ALL' &BLANKS '*YES' +
                          &ERRCDE '*USER')
/*                                                                   */
/* Check for errors on QUSCRTUS                                      */
/*                                                                   */
             IF         COND(&BYTAVL *GT 0) THEN(DO)
             SNDPGMMSG  MSG('Failure with QUSCRTUS') TOPGMQ(*EXT)
             RETURN
             ENDDO
/*                                                                   */
/* Else an error accessing the User Space                            */
/*                                                                   */
             ELSE       CMD(DO)
             SNDPGMMSG  MSG('Failure with QUSROBJD') TOPGMQ(*EXT)
             RETURN
             ENDDO
             ENDDO
             ENDDO
/*                                                                   */
/* Set QSYLOBJP (via GETLST) to start a new list                     */
/*                                                                   */
             CHGVAR     VAR(&CONTIN_HDL) VALUE(&BLANKS)
             CALLSUBR   SUBR(GETLST)
/*                                                                   */
/* Get a resolved pointer to the User Space                          */
/*                                                                   */
             CALL       PGM(QUSPTRUS) PARM(&SPC_NAME &SPCPTR &ERRCDE)
/*                                                                   */
/* Check for errors on QUSPTRUS                                      */
/*                                                                   */
             IF         COND(&BYTAVL *GT 0) THEN(DO)
             SNDPGMMSG  MSG('Failure with QUSPTRUS') TOPGMQ(*EXT)
```

```
            RETURN
            ENDDO
            ENDSUBR
            ENDPGM
```

Example in ILE C: List APIs

This ILE C program prints a report that shows all objects that adopt owner authority.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* Program:      List objects that adopt owner authority             */
/*                                                                   */
/* Language:     ILE C                                               */
/*                                                                   */
/* Description:  This program prints a report showing all objects    */
/*               that adopt owner authority.  The two parameters     */
/*               passed to the program are the profile to be         */
/*               checked and the type of objects to be listed.       */
/*               The parameter values are the same as those          */
/*               accepted by the QSYLOBJP API.                       */
/*                                                                   */
/* APIs Used:    QSYLOBJP - List Objects that Adopt Owner Authority  */
/*               QUSCRTUS - Create User Space                        */
/*               QUSPTRUS - Retrieve Pointer to User Space           */
/*               QUSROBJD - Retrieve Object Description              */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <stdio.h>
#include <string.h>
#include <qsylobjp.h>    /* QSYLOBJP API Header                      */
#include <quscrtus.h>    /* QUSCRTUS API Header                      */
#include <qusptrus.h>    /* QUSPTRUS API Header                      */
#include <qusrobjd.h>    /* QUSROBJD API Header                      */
#include <qusgen.h>      /* Format Structures for User Space         */
#include <qusec.h>       /* Error Code Parameter Include for the APIs */
#include <qliept.h>      /* Entry Point Table Include                */

/*********************************************************************/
/* Error Code Structure                                              */
/*                                                                   */
/* This shows how the user can define the variable length portion of  */
/* error code for the exception data.                                */
/*                                                                   */
/*********************************************************************/
typedef struct {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
        } error_code_t;

/*********************************************************************/
/* Global Variables                                                  */
/*********************************************************************/
char       api_name[10];
char       cont_hdl[20];
char       ext_attr[10];
char       list_status;
char       mbr_list[8];
char       obj_type[10];
char       rcvvar[8];
char       rjobd_fmt[8];
char       space_auth[10];
```

```
char       space_dmn[10];
char       space_init;
char       space_name[20];
char       space_rep[10];
char       space_text[50];
char       space_type[10];
char       usr_prf[10];
char       *usrspc_ptr, *usrspc_base;
int        rcvlen = 8;
int        size_entry;
int        space_size = 1;
error_code_t error_code;
FILE       *record;

/*********************************************************************/
/* Function:      done                                               */
/*                                                                   */
/* Description:   This function prints the end of listing print line */
/*                and returns to the caller.                         */
/*********************************************************************/
void done()
{
  char command_string[32];

  fwrite("*** End of List",1, 15, record);
  fclose(record);
  exit();

} /* done */

/*********************************************************************/
/* Function:      apierr                                             */
/*                                                                   */
/* Description:   This function prints the API name, and exception   */
/*                identifier of an error that occurred.              */
/*********************************************************************/
void apierr()
{
  printf("API: %.10s\n", api_name);
  printf("Failed with exception: %.7s\n",
          error_code.ec_fields.Exception_Id);
  done();

} /* apierr */


/*********************************************************************/
/* Function:      getlst                                             */
/*                                                                   */
/* Description:   This function calls QSYLOBJP to build a list.     */
/*                                                                   */
/*********************************************************************/
void getlst()
{
  memcpy(mbr_list, "OBJP0200", 8);

  /*******************************************************************/
  /* Call QSYLOBJP API to generate a list.  The continuation handle  */
  /* is set by the caller of this function.                          */
  /*******************************************************************/
  QSYLOBJP(space_name,             /* User space and library         */
           mbr_list,               /* Member list                    */
           usr_prf,                /* User profile                   */
           obj_type,               /* Object type                    */
           cont_hdl,               /* Continuation handle    (3)     */
           &error_code);           /* Error code                     */
```

```
  /*****************************************************************/
  /* Check for errors on QSYLOBJP.                                 */
  /*****************************************************************/
  if(error_code.ec_fields.Bytes_Available > 0)
  {
    memcpy(api_name, "QSYLOBJP  ", 10);
    apierr();
  }

} /* getlst */

/*********************************************************************/
/* Function:      init                                               */
/*                                                                   */
/* Description:   This function does all the necessary initialization */
/*                for this program.                                  */
/*********************************************************************/
void init()
{
  memcpy(space_name, "ADOPTS    QTEMP     ", 20);
  space_init = 0x00;
  memcpy(mbr_list, "OBJP0200", 8);
  memcpy(rjobd_fmt, "OBJD0100", 8);
  memcpy(space_type, "*USRSPC   ", 10);
  memcpy(ext_attr, "QSYLOBJP  ", 10);
  memcpy(space_auth, "*ALL      ", 10);
  memcpy(space_rep, "*YES      ", 10);
  memcpy(space_dmn, "*USER     ", 10);

  /*****************************************************************/
  /* Open QPRINT file so that data can be written to it.  If the file */
  /* cannot be opened, print a message and exit.                   */
  /*****************************************************************/
  if((record = fopen("QPRINT", "wb, lrecl=132, type=record")) == NULL)
  {
    printf("File could not be opened\n");
    exit(1);
  }

  error_code.ec_fields.Bytes_Provided = sizeof(error_code_t);

  /*****************************************************************/
  /* Call QUSROBJD to see if the user space was previously created in */
  /* QTEMP.  If it was, simply reuse it.                           */
  /*****************************************************************/
  QUSROBJD(rcvvar,                 /* Receiver variable            */
           rcvlen,                 /* Receiver variable length     */
           rjobd_fmt,              /* Format                       */
           space_name,             /* User space name and library  */
           space_type,             /* User object type             */
           &error_code);           /* Error code                   */

  if(error_code.ec_fields.Bytes_Available > 0)
  {
    /***************************************************************/
    /* If a CPF9801 error was received, then the user space was not  */
    /* found.                                                      */
    /***************************************************************/
    if(memcmp(error_code.ec_fields.Exception_Id, "CPF9801", 7) == 0)
    {
      /*************************************************************/
      /* Create a user space for the list generated by QSYLOBJP.   */
      /*************************************************************/
      QUSCRTUS(space_name,         /* User space name and library  */
               ext_attr,           /* Extended attribute           */
               space_size,         /* Size of the user space       */
               &space_init,        /* Space initialization         */
```

```
                space_auth,          /* Public authority to user space */
                space_text,          /* User space text                */
                space_rep,           /* Replace existing user space?   */
                &error_code,         /* Error Code                     */
                space_dmn);          /* Domain of created user space   */

      /*****************************************************************/
      /* Check for errors on QUSCRTUS.                                 */
      /*****************************************************************/
      if(error_code.ec_fields.Bytes_Available > 0)
      {
         memcpy(api_name, "QUSCRTUS  ", 10);
         apierr();
      }
    }
    /*******************************************************************/
    /* An error occurred accessing the user space.                     */
    /*******************************************************************/
    else
    {
      memcpy(api_name, "QUSRJOBD  ", 10);
      apierr();
    }
  }

    /*******************************************************************/
    /* Set QSYLOBJP (via GETLST) to start a new list.                  */
    /*******************************************************************/
    memset(cont_hdl, ' ', 20);
    getlst();

    /*******************************************************************/
    /* Get a resolved pointer to the user space for performance.       */
    /*******************************************************************/
    QUSPTRUS(space_name,             /* User space name and library    */
             &usrspc_ptr,            /* User space pointer             */
             &error_code);           /* Error Code                     */

    /*******************************************************************/
    /* Check for errors on QUSPTRUS.                                   */
    /*******************************************************************/
    if(error_code.ec_fields.Bytes_Available > 0)
    {
      memcpy(api_name, "QUSPTRUS  ", 10);
      apierr();
    }

    usrspc_base = usrspc_ptr;

} /* init */

/*********************************************************************/
/* Function:      proces2                                            */
/*                                                                   */
/* Description:   This function processes each entry returned by     */
/*                QSYLOBJP.                                          */
/*                                                                   */
/*********************************************************************/
void proces2()
{
  char obj_type[112];

  sprintf(obj_type, "Object: %.10s Library: %.10s Type: %.10s Text: %.50s\n",
         ((Qsy_OBJP0200_List_T *)usrspc_ptr)->Object.Name,
         ((Qsy_OBJP0200_List_T *)usrspc_ptr)->Object.Library,
         ((Qsy_OBJP0200_List_T *)usrspc_ptr)->Object_Type,
         ((Qsy_OBJP0200_List_T *)usrspc_ptr)->Object_Text);
```

```
  fwrite(obj_type, 1, 112, record);

  /*******************************************************************/
  /* After each entry, increment to the next entry.                  */
  /*******************************************************************/
  usrspc_ptr += size_entry;                                 (7)

} /* proces2 */

/*********************************************************************/
/* Function:      proces1                                            */
/*                                                                   */
/* Description:   This function processes each entry returned by     */
/*                QSYLOBJP.                                          */
/*                                                                   */
/*********************************************************************/
void proces1()
{
  int i;
  int num_entries;
  int offset;

  num_entries = ((Qus_Generic_Header_0100_t *)\
                 usrspc_ptr)->Number_List_Entries;

  /*******************************************************************/
  /* If valid information was returned.      (1)                     */
  /*******************************************************************/
  if((((Qus_Generic_Header_0100_t *)usrspc_ptr)->Information_Status == 'C') ||
     (((Qus_Generic_Header_0100_t *)usrspc_ptr)->Information_Status == 'P'))
  {
    if(num_entries > 0)
    {
      /*****************************************************************/
      /* Get the size of each entry to use later.   (4)                */
      /*****************************************************************/
      size_entry = ((Qus_Generic_Header_0100_t *)usrspc_ptr)->Size_Each_Entry;

      /*****************************************************************/
      /* Increment to the first list entry.                            */
      /*****************************************************************/
      offset = ((Qus_Generic_Header_0100_t *)usrspc_ptr)->Offset_List_Data;  (5)
      usrspc_ptr += offset;

      /*****************************************************************/
      /* Process all of the entries.                                   */
      /*****************************************************************/
      for(i=0; i<num_entries; i++)                          (6)
        proces2();

      /*****************************************************************/
      /* Reset the user space pointer to the beginning.                */
      /*****************************************************************/
      usrspc_ptr = usrspc_base;

      /*****************************************************************/
      /* If all entries in this user space have been processed, check */
      /* if more entries exist than can fit in one user space.         */
      /*****************************************************************/
      if(((Qus_Generic_Header_0100_t *)usrspc_ptr)->Information_Status == 'P')
      {
        /***************************************************************/
        /* Address the input parameter header.                         */
        /***************************************************************/
        offset = ((Qus_Generic_Header_0100_t *)\
                   usrspc_ptr)->Offset_Input_Parameter;
        usrspc_ptr += offset;
```

```
        /************************************************************/
        /* If the continuation handle in the input parameter header   */
        /* is blank, then set the list status to complete.            */
        /************************************************************/
        if(memcmp(((Qsy_OBJP_Input_T *)usrspc_ptr)->Continuation_Handle,
           "                    ", 20) == 0)
        {
          list_status = 'C';
        }
        else
        /************************************************************/
        /* Else, call QSYLOBJP reusing the user space to get more     */
        /* list entries.                                              */
        /************************************************************/
        {
          memcpy(cont_hdl, ((Qsy_OBJP_Input_T *)\
                 usrspc_ptr)->Continuation_Handle, 20);     (2)
          getlst();
          list_status = ((Qus_Generic_Header_0100_t *)\
                          usrspc_ptr)->Information_Status;
        }
      }
    }
    else
    /****************************************************************/
    /* If there exists an unexpected status, log an error (not shown) */
    /* and exit.                                                    */
    /****************************************************************/
    {
      done();
    }
  }
} /* proces1 */

/********************************************************************/
/* Function:      proces                                            */
/*                                                                  */
/* Description:   Processes entries until they are complete.        */
/*                                                                  */
/********************************************************************/
void proces()
{
  list_status = ((Qus_Generic_Header_0100_t *)usrspc_ptr)->Information_Status;

  do
  {
    proces1();
  } while (list_status != 'C');

} /* proces */

/********************************************************************/
/* main                                                             */
/********************************************************************/

main(int argc, char *argv[])
{
  /******************************************************************/
  /* Make sure we received the correct number of parameters. The argc */
  /* parameter will contain the number of parameters that was passed  */
  /* to this program. This number also includes the program itself,   */
  /* so we need to evaluate argc-1.                                   */
  /******************************************************************/

  if (((argc - 1) < 2) || ((argc - 1 > 2)))
  /******************************************************************/
```

```
  /* We did not receive all of the required parameters so exit the    */
  /* program.                                                         */
  /*******************************************************************/
  {
    exit(1);
  }
  else
  /*******************************************************************/
  /* Copy parameters into local variables.                           */
  /*******************************************************************/
  {
    memcpy(usr_prf, argv[1], 10);
    memcpy(obj_type, argv[2], 10);
  }

  init();
  proces();
  done();

} /* main */
```

Related reference:

"Example in OPM RPG: List APIs" on page 87
This OPM RPG program prints a report that shows all objects that adopt owner authority.

Example in ILE RPG: List APIs

This ILE RPG program prints a report that shows all objects that adopt owner authority.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
     F*****************************************************************
     F*****************************************************************
     F*
     F*  Program:      List objects that adopt owner authority
     F*
     F*  Language:     ILE RPG
     F*
     F*  Description:  This program prints a report showing all objects
     F*                that adopt owner authority.  The two parameters
     F*                passed to the program are the profile to be
     F*                checked and the type of objects to be listed.
     F*                The parameter values are the same as those
     F*                accepted by the QSYLOBJP API.
     F*
     F*  APIs Used:    QSYLOBJP - List Objects that Adopt Owner Authority
     F*                QUSCRTUS - Create User Space
     F*                QUSPTRUS - Retrieve Pointer to User Space
     F*                QUSROBJD - Retrieve Object Description
     F*
     F*****************************************************************
     F*****************************************************************
     F*
     FQPRINT    O    F  132        PRINTER OFLIND(*INOF)
     D*
     D* Error Code parameter include
     D*
     D/COPY QSYSINC/QRPGLESRC,QUSEC
     D*
     DSPC_NAME         S             20    INZ('ADOPTS    QTEMP     ')
     DSPC_SIZE         S              9B 0 INZ(1)
     DSPC_INIT         S              1    INZ(X'00')
     DLSTPTR           S               *
     DSPCPTR           S               *
     DARR              S              1    BASED(LSTPTR) DIM(32767)
     DRCVVAR           S              8
```

```
DRCVVARSIZ        S              9B 0 INZ(8)
D*****************************************************************
D*
D* The following QUSGEN include from QSYSINC is copied into
D* this program so that it can be declared as BASED on SPCPTR
D*
D*****************************************************************
DQUSH0100         DS                  BASED(SPCPTR)
D*                                            Qus Generic Header 0100
D QUSUA                   1     64
D*                                            User Area
D QUSSGH                 65     68B 0
D*                                            Size Generic Header
D QUSSRL                 69     72
D*                                            Structure Release Level
D QUSFN                  73     80
D*                                            Format Name
D QUSAU                  81     90
D*                                            API Used
D QUSDTC                 91    103
D*                                            Date Time Created
D QUSIS                 104    104
D*                                            Information Status
D QUSSUS                105    108B 0
D*                                            Size User Space
D QUSOIP                109    112B 0
D*                                            Offset Input Parameter
D QUSSIP                113    116B 0
D*                                            Size Input Parameter
D QUSOHS                117    120B 0
D*                                            Offset Header Section
D QUSSHS                121    124B 0
D*                                            Size Header Section
D QUSOLD                125    128B 0
D*                                            Offset List Data
D QUSSLD                129    132B 0
D*                                            Size List Data
D QUSNBRLE              133    136B 0
D*                                            Number List Entries
D QUSSEE                137    140B 0
D*                                            Size Each Entry
D QUSSIDLE              141    144B 0
D*                                            CCSID List Ent
D QUSCID                145    146
D*                                            Country ID
D QUSLID                147    149
D*                                            Language ID
D QUSSLI                150    150
D*                                            Subset List Indicator
D QUSERVED00            151    192
D*                                            Reserved
D*****************************************************************
D*
D* The following QSYLOBJP include from QSYSINC is copied into
D* this program so that it can be declared as BASED on LSTPTR
D*
D*****************************************************************
D QSYLOBJP        C                   'QSYLOBJP'
D*****************************************************************
D*Header structure for QSYLOBJP
D*****************************************************************
DQSYOBJPH         DS                  BASED(LSTPTR)
D*                                            Qsy OBJP Header
D QSYUN00                 1     10
D*                                  User name
D QSYCV00                11     30
D*                                  Continuation Value
```

```
D*****************************************************************
D*Record structure for OBJP0200 format
D*****************************************************************
DQSY0200L02       DS                  BASED(LSTPTR)
D*                                             Qsy OBJP0200 List
D  QSYNAME06              1     10
D*                                             Name
D  QSYBRARY06            11     20
D*                                             Library
D QSYOBJT13               21     30
D*                                             Object Type
D QSYOBJIU00              31     31
D*                                             Object In Use
D QSYOBJA11               32     41
D*                                             Object Attribute
D QSYOBJT14               42     91
D*                                             Object Text
C*
C* Start of mainline
C*
C     *ENTRY        PLIST
C                   PARM                    USR_PRF          10
C                   PARM                    OBJ_TYPE         10
C                   EXSR      INIT
C                   EXSR      PROCES
C                   EXSR      DONE
C*
C* Start of subroutines
C*
C*****************************************************************
C     PROCES        BEGSR
C*
C* This subroutine processes each entry returned by QSYLOBJP
C*
C*
C* Do until the list is complete
C*
C                   MOVE      QUSIS         LST_STATUS        1
C*
C     LST_STATUS    DOUEQ     'C'
C*
C* If valid information was returned
C*
C     QUSIS         IFEQ      'C'
C     QUSIS         OREQ      'P'
C*
C* and list entries were found
C*
C     QUSNBRLE      IFGT      0
C*
C* set LSTPTR to the first byte of the User Space
C*
C                   EVAL      LSTPTR = SPCPTR
C*
C* increment LSTPTR to the first List entry
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSOLD + 1))  (5)
C*
C* and process all of the entries
C*
C                   DO        QUSNBRLE                 (6)
C                   EXCEPT    OBJ_ENTRY
C*
C* after each entry, increment LSTPTR to the next entry
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSSEE + 1))  (7)
C                   END
```

```
C                   END
C*
C* If all entries in this User Space have been processed, check
C* if more entries exist than can fit in one User Space
C*
C     QUSIS         IFEQ      'P'
C*
C* by resetting LSTPTR to the start of the User Space
C*
C                   EVAL      LSTPTR = SPCPTR
C*
C* and then incrementing LSTPTR to the Input Parameter Header
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSOIP + 1))
C*
C* If the continuation handle in the Input Parameter Header is
C* blank, then set the List status to Complete
C*
C     QSYCV00       IFEQ      *BLANKS
C                   MOVE      'C'           LST_STATUS
C                   ELSE
C*
C* Else, call QSYLOBJP reusing the User Space to get more
C* List entries
C*
C                   MOVE      QSYCV00       CONTIN_HDL   (2)
C                   EXSR      GETLST
C                   MOVE      QUSIS         LST_STATUS
C                   END
C                   END
C                   ELSE
C*
C* And if an unexpected status, log an error (not shown) and exit
C*
C                   EXSR      DONE
C                   END
C                   END
C                   ENDSR
C*****************************************************************
C     GETLST        BEGSR
C*
C* Call QSYLOBJP to generate a list
C* The continuation handle is set by the caller of this
C* subroutine.
C*
C                   CALL      QSYLOBJP
C                   PARM                    SPC_NAME
C                   PARM      'OBJP0200'    MBR_LIST          8
C                   PARM                    USR_PRF
C                   PARM                    OBJ_TYPE
C                   PARM                    CONTIN_HDL       20  (3)
C                   PARM                    QUSEC
C*
C* Check for errors on QSYLOBJP
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QSYLOBJP'    APINAM           10
C                   EXSR      APIERR
C                   END
C                   ENDSR
C*****************************************************************
C     INIT          BEGSR
C*
C* One time initialization code for this program
C*
C* Set Error Code structure not to use exceptions
C*
```

```
C                   Z-ADD     16            QUSBPRV
C*
C* Check to see if the User Space was previously created in
C* QTEMP.  If it was, simply reuse it.
C*
C                   CALL      'QUSROBJD'
C                   PARM                    RCVVAR
C                   PARM                    RCVVARSIZ
C                   PARM      'OBJD0100'    ROBJD_FMT         8
C                   PARM                    SPC_NAME
C                   PARM      '*USRSPC'     SPC_TYPE         10
C                   PARM                    QUSEC
C*
C* Check for errors on QUSROBJD
C*
C     QUSBAVL       IFGT      0
C*
C* If CPF9801, then User Space was not found
C*
C     QUSEI         IFEQ      'CPF9801'
C*
C* So create a User Space for the List generated by QSYLOBJP
C*
C                   CALL      'QUSCRTUS'
C                   PARM                    SPC_NAME
C                   PARM      'QSYLOBJP  '  EXT_ATTR         10
C                   PARM                    SPC_SIZE
C                   PARM                    SPC_INIT
C                   PARM      '*ALL'        SPC_AUT          10
C                   PARM      *BLANKS       SPC_TEXT         50
C                   PARM      '*YES'        SPC_REPLAC       10
C                   PARM                    QUSEC
C                   PARM      '*USER'       SPC_DOMAIN       10
C*
C* Check for errors on QUSCRTUS
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QUSCRTUS'    APINAM           10
C                   EXSR      APIERR
C                   END
C*
C* Else, an error occurred accessing the User Space
C*
C                   ELSE
C                   MOVEL     'QUSROBJD'    APINAM           10
C                   EXSR      APIERR
C                   END
C                   END
C*
C* Set QSYLOBJP (via GETLST) to start a new list
C*
C                   MOVE      *BLANKS       CONTIN_HDL
C                   EXSR      GETLST
C*
C* Get a resolved pointer to the User Space for performance
C*
C                   CALL      'QUSPTRUS'
C                   PARM                    SPC_NAME
C                   PARM                    SPCPTR
C                   PARM                    QUSEC
C*
C* Check for errors on QUSPTRUS
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QUSPTRUS'    APINAM           10
C                   EXSR      APIERR
C                   END
```

```
C                   ENDSR
C*****************************************************************
C     APIERR        BEGSR
C*
C* Log any error encountered, and exit the program
C*
C     APINAM        DSPLY
C     QUSEI         DSPLY
C                   EXSR      DONE
C                   ENDSR
C*****************************************************************
C     DONE          BEGSR
C*
C* Exit the program
C*
C                   EXCEPT    END_LIST
C                   EVAL      *INLR = '1'
C                   RETURN
C                   ENDSR
OQPRINT    E            OBJ_ENTRY      1
O                                              'Object: '
O                       QSYNAME06
O                                              ' Library: '
O                       QSYBRARY06
O                                              ' Type: '
O                       QSYOBJT13
O                                              ' Text: '
O                       QSYOBJT14
OQPRINT    E            END_LIST       1
O                                              '*** End of List'
```

Related reference:

"Example in OPM RPG: List APIs" on page 87
This OPM RPG program prints a report that shows all objects that adopt owner authority.

Example in ILE COBOL: List APIs

This ILE COBOL program prints a report that shows all objects that adopt owner authority.

The following program also works for OPM COBOL.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*  Program:      List objects that adopt owner authority
*
*  Language:     COBOL
*
*  Description:  This program prints a report showing all objects
*                that adopt owner authority.  The two parameters
*                passed to the program are the profile to be
*                checked and the type of objects to be listed.
*                The parameter values are the same as those
*                accepted by the QSYLOBJP API.
*
*  APIs Used:    QSYLOBJP - List Objects that Adopt Owner Authority
*                QUSCRTUS - Create User Space
*                QUSPTRUS - Retrieve Pointer to User Space
*                QUSROBJD - Retrieve Object Description
*
*****************************************************************
*****************************************************************
*
```

```
 PROGRAM-ID. LISTADOPT.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION.
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
*
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Listing text
*
 01  OBJ-ENTRY.
     05  OBJECT-FIELD.
         09  TEXT1         PIC  X(08) VALUE "Object: ".
         09  NAME          PIC  X(10).
         09  TEXT2         PIC  X(10) VALUE " Library: ".
         09  LIBRARY-FIELD       PIC  X(10).
     05  TEXT3         PIC  X(07) VALUE " Type: ".
     05  OBJECT-TYPE   PIC  X(10).
     05  TEXT4         PIC  X(07) VALUE " Text: ".
     05  OBJECT-TEXT   PIC  X(50).
 01  END-LIST.
     05  TEXT1         PIC  X(15) VALUE "*** End of List".
*
 01  MISC.
     05  SPC-NAME      PIC  X(20) VALUE "ADOPTS    QTEMP     ".
     05  SPC-SIZE      PIC S9(09) VALUE 1 BINARY.
     05  SPC-INIT      PIC  X(01) VALUE X"00".
     05  SPCPTR        POINTER.
     05  RCVVAR        PIC  X(08).
     05  RCVVARSIZ     PIC S9(09) VALUE 8 BINARY.
     05  LST-STATUS    PIC  X(01).
     05  MBR-LIST      PIC  X(08) VALUE "OBJP0200".
     05  CONTIN-HDL    PIC  X(20).
     05  APINAM        PIC  X(10).
     05  ROBJD-FMT     PIC  X(08) VALUE "OBJD0100".
     05  SPC-TYPE      PIC  X(10) VALUE "*USRSPC".
     05  EXT-ATTR      PIC  X(10) VALUE "QSYLOBJP".
     05  SPC-AUT       PIC  X(10) VALUE "*ALL".
     05  SPC-TEXT      PIC  X(50).
     05  SPC-REPLAC    PIC  X(10) VALUE "*YES".
     05  SPC-DOMAIN    PIC  X(10) VALUE "*USER".
*
 LINKAGE SECTION.
*
* Input parameters.
*
 01  USR-PRF         PIC  X(10).
 01  OBJ-TYPE        PIC  X(10).
*
* String to map User Space offsets into
*
```

```
 01  STRING-SPACE    PIC  X(32000).
*
* User Space Generic Header include.  These includes will be
* mapped over a User Space.
*
 COPY QUSGEN OF QSYSINC-QLBLSRC.
*
* List Objects that Adopt API include.  These includes will be
* mapped over a User Space.
*
 COPY QSYLOBJP OF QSYSINC-QLBLSRC.
*
* Beginning of mainline
*
 PROCEDURE DIVISION USING USR-PRF, OBJ-TYPE.
 MAIN-LINE.
     PERFORM INIT.
     PERFORM PROCES.
     PERFORM DONE.
*
* Start of subroutines
*
*****************************************************************
 PROCES.
*
* Do until the list is complete
*
     MOVE INFORMATION-STATUS OF QUS-GENERIC-HEADER-0100 TO
                                LST-STATUS.
*
     PERFORM PROCES1 WITH TEST AFTER UNTIL LST-STATUS = "C".
*
 PROCES1.
*
* This subroutine processes each entry returned by QSYLOBJP
*
*
* If valid information was returned
*
     IF (INFORMATION-STATUS OF QUS-GENERIC-HEADER-0100 = "C"
        OR INFORMATION-STATUS OF QUS-GENERIC-HEADER-0100 = "P")
      IF NUMBER-LIST-ENTRIES OF QUS-GENERIC-HEADER-0100 > 0
*
* increment to the first list entry
*
        SET ADDRESS OF QSY-OBJP0200-LIST TO
           ADDRESS OF STRING-SPACE(
           (OFFSET-LIST-DATA OF QUS-GENERIC-HEADER-0100 + 1):1),  (5)
        SET ADDRESS OF STRING-SPACE TO ADDRESS OF
           QSY-OBJP0200-LIST,
*
* and process all of the entries
*
        PERFORM PROCES2
           NUMBER-LIST-ENTRIES OF QUS-GENERIC-HEADER-0100 TIMES,  (6)
*
* If all entries in this User Space have been processed, check
* if more entries exist than can fit in one User Space
*
        IF INFORMATION-STATUS OF QUS-GENERIC-HEADER-0100 = "P"
*
* by addressing the input parameter header
*
           SET ADDRESS OF STRING-SPACE TO SPCPTR,
           SET ADDRESS OF QSY-OBJP-INPUT TO
              ADDRESS OF STRING-SPACE((OFFSET-INPUT-PARAMETER
              OF QUS-GENERIC-HEADER-0100 + 1):1),
```

```
*
* If the continuation handle in the Input Parameter Header is
* blank, then set the List status to Complete
*
           IF CONTINUATION-HANDLE OF QSY-OBJP-INPUT = SPACES
              MOVE "C" TO LST-STATUS
              ELSE
*
* Else, call QSYLOBJP reusing the User Space to get more
* List entries
*
              MOVE CONTINUATION-HANDLE OF QSY-OBJP-INPUT
                   TO CONTIN-HDL OF MISC,             (2)
              PERFORM GETLST,
              MOVE INFORMATION-STATUS OF QUS-GENERIC-HEADER-0100
                   TO LST-STATUS,
              END-IF,
        END-IF,
      END-IF,
     ELSE
*
* And if an unexpected status, log an error (not shown) and exit
*
     PERFORM DONE,
     END-IF.
*
 PROCES2.
     MOVE CORRESPONDING QSY-OBJP0200-LIST TO OBJ-ENTRY.
     WRITE LIST-LINE FROM OBJ-ENTRY.
*
* after each entry, increment to the next entry
*
     SET ADDRESS OF QSY-OBJP0200-LIST TO ADDRESS OF
        STRING-SPACE(
         (SIZE-EACH-ENTRY OF QUS-GENERIC-HEADER-0100 + 1):1).  (7)
     SET ADDRESS OF STRING-SPACE TO ADDRESS OF QSY-OBJP0200-LIST.
*****************************************************************
 GETLST.
*
* Call QSYLOBJP to generate a list
* The continuation handle is set by the caller of this
* subroutine.
     MOVE "OBJP0200" TO MBR-LIST.
*
     CALL "QSYLOBJP" USING SPC-NAME, MBR-LIST, USR-PRF,
                           OBJ-TYPE, CONTIN-HDL, QUS-EC.  (3)
*
* Check for errors on QSYLOBJP
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           MOVE "QSYLOBJP" TO APINAM,
                           PERFORM APIERR.
*****************************************************************
 INIT.
*
* One time initialization code for this program
*
* Open LISTING file
*
     OPEN OUTPUT LISTING.
*
* Set Error Code structure to not use exceptions
*
     MOVE LENGTH OF QUS-EC TO BYTES-PROVIDED OF QUS-EC.
*
* Check to see if the User Space was previously created in
* QTEMP.  If it was, simply reuse it.
```

```
*
     CALL "QUSROBJD" USING RCVVAR, RCVVARSIZ, ROBJD-FMT,
                           SPC-NAME, SPC-TYPE, QUS-EC.
*
* Check for errors on QUSROBJD
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
*
* If CPF9801, then User Space was not found
*
        IF EXCEPTION-ID OF QUS-EC = "CPF9801"
*
* So create a User Space for the List generated by QSYLOBJP
*
           CALL "QUSCRTUS" USING SPC-NAME, EXT-ATTR, SPC-SIZE,
                           SPC-INIT, SPC-AUT, SPC-TEXT,
                           SPC-REPLAC, QUS-EC, SPC-DOMAIN
*
* Check for errors on QUSCRTUS
*
           IF BYTES-AVAILABLE OF QUS-EC > 0
                             MOVE "QUSCRTUS" TO APINAM,
                             PERFORM APIERR,
                             ELSE
                             CONTINUE,
        ELSE
*
* Else, an error occurred accessing the User Space
*
           MOVE "QUSROBJD" TO APINAM,
           PERFORM APIERR.
*
* Set QSYLOBJP (via GETLST) to start a new list
*
     MOVE SPACES TO CONTIN-HDL.
     PERFORM GETLST.
*
* Get a resolved pointer to the User Space for performance
*
     CALL "QUSPTRUS" USING SPC-NAME, SPCPTR, QUS-EC.
*
* Check for errors on QUSPTRUS
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           MOVE "QUSPTRUS" TO APINAM,
                           PERFORM APIERR.
*
* If no error, then set addressability to User Space
*
     SET ADDRESS OF QUS-GENERIC-HEADER-0100 TO SPCPTR.
     SET ADDRESS OF STRING-SPACE TO SPCPTR.
*
****************************************************************
 APIERR.
*
* Log any error encountered, and exit the program
*
     DISPLAY APINAM.
     DISPLAY EXCEPTION-ID OF QUS-EC.
     PERFORM DONE.
****************************************************************
 DONE.
*
* Exit the program
*
     WRITE LIST-LINE FROM END-LIST.
     STOP RUN.
```

Related reference:

"Example in OPM RPG: List APIs" on page 87
This OPM RPG program prints a report that shows all objects that adopt owner authority.

Domains

A *domain* is a characteristic of an object that controls how programs can access the object. All objects are assigned a domain attribute when they are created.

After a domain is set, it remains in effect for the life of the object. The possible attributes are *system* and *user*.

Most object types on the system are created in system domain. When you run your system at security level 40 or 50, you can access system domain objects only by using the commands and callable APIs provided.

The following object types can be either system or user domain. The list includes the symbolic object type.

- User space (*USRSPC)
- User index (*USRIDX)
- User queue (*USRQ)

Objects of the type *USRSPC, *USRIDX, and *USRQ in the user domain can be manipulated directly by MI instructions without using the system-provided APIs and commands.

Note: Objects of the type *PGM, *SRVPGM, and *SQLPKG can also be in the user domain. Their contents cannot be manipulated directly by MI instructions.

User objects can exist in either the user domain or the system domain. The QALWUSRDMN system value determines which libraries can contain user-domain user objects. The default QALWUSRDMN system value is set to *ALL, but it can be changed by system administrators on individual systems to be one library or a list of libraries. If your application requires direct pointer access to user-domain user objects in a library that is not specified in the QALWUSRDMN system value, your system administrator can add the library to the system value.

The ability to create user domain objects on a system with a security level 40 or 50 is controlled by the QALWUSRDMN system value. For more information, see the User queue domain table in Create User Queue (QUSCRTUQ) API.

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

Related reference:

Security reference

"Examples: Using data queues or user queues" on page 282
Both data queues and user queues provide a means for one or more processes to communicate asynchronously. Both queues can be processed by first-in first-out (FIFO), last-in first-out (LIFO), or by key.

Exit programs

An *exit program* is a program to which control is passed from a calling application program or system program. Exit programs can be used to customize particular functions to your needs.

Exit programs are usually user-written programs; however, a few are system-supplied (such as a few of the Operational Assistant exit programs).

To transfer control to an exit program, you do an external call as you would to any other program.

There are no general requirements for using exit programs. For any specific requirements, see the documentation for the specific exit program.

Exit points

An *exit point* signifies the point in a system function or program where control is turned over to one or more exit programs to perform a function.

The *registration facility* provides a central point to store and retrieve information about IBM i and non-IBM i exit points and their associated exit programs. This information is stored in the registration facility repository and can be retrieved to determine which exit points and exit programs already exist.

You can use the registration facility APIs to register and deregister exit points, to add and remove exit programs, and to retrieve information about exit points and exit programs. You can also perform some of these functions by using the Work with Registration Information (WRKREGINF) command.

The *exit point provider* is responsible for defining the exit point information, defining the format in which the exit program receives data, and calling the exit program.

Related reference:

Work with Registration Information (WRKREGINF) command

User index considerations

In IBM i APIs, a *user index* is an object that provides a specific order for byte data according to the value of the data. A user index has better performance than a database file. However, before using a user index, you must know its functional differences from a database file.

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

If your system abnormally ends when you are removing or inserting a user index entry, unpredictable results might occur. If you are inserting or removing a user index entry, you need to force the index entry to the disk unit using one of the following:

- A user index created with the immediate update parameter set to 1 (affects performance)
- A Modify Independent Index (MODIDX) MI instruction with the immediate update bit set to 1
- The Set Access State (SETACST) MI instruction

If you do not force the index entry and the system abnormally ends, your index is probably damaged.

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

If your index is damaged, you do not get an error message. The definition of your index is usable; it is probably the data in your index that is bad.

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

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

Related reference:

Journal and Commit APIs

Performance considerations

The format specified for an API influences the performance cost of the API. In general, when more information is returned, the performance is slower.

Some list APIs, such as List Job (QUSLJOB), List Spooled Files (QUSLSPL), and List Objects (QUSLOBJ), generate the list with minimal cost. This is why the formats specified for these APIs do not retrieve very much information. Some of the APIs, such as List Record Formats (QUSLRCD) and List Fields (QUSLFLD), have only one format, because there is no additional performance cost to supply the complete information.

The retrieve APIs allow you to control the performance cost for information that you retrieve. The retrieve APIs, such as Retrieve Member Description (QUSRMBRD) and Retrieve Spooled File Attributes (QUSRSPLA), have formats that are generally ordered from the fastest performance to the slowest performance. That is, the lower-numbered formats run faster but retrieve less information, and the higher-numbered formats run slower but retrieve more information. One exception is the Retrieve Job Information (QUSRJOBI) API, where the order of the formats does not have anything to do with performance characteristics.

Related reference:

Retrieve Job Information (QUSRJOBI) API

APIs and system objects

APIs retrieve information from system objects.

Some of the returned information contains special values. For example, the List Objects (QUSLOBJ) API returns the object type as a special value (such as *PGM and *LIB). However, special values might be added in future releases. Even numeric values might have new special values. When you code to APIs, assume that the format of the information returned does not change from release to release, but the content of the information might change.

Open list information format

The format of the open list information is common across many of the open list APIs.

This common open list structure provides information necessary for the API caller to properly process the list. If the API error code parameter indicates that no error occurred, the information complete indicator should be checked for a value of either C (complete and accurate) or P (partial and accurate). If one of these values is found, the API caller should process the number of entries indicated by the records returned field.

When these records have been processed, the API caller should determine whether all records that can be returned in the list have been returned. The API caller can determine this by comparing the total records value with the sum of the first record in receiver variable value and the records returned value less 1. When the total records value is greater than or equal to the first record in receiver variable value plus the records returned value minus 1, additional calls to the Get List Entries (QGYGTLE) API can continue to receive new records if the list status indicator is not 3 or 5. When total records have been processed and the list status indicator is 2 or 5, or if the caller no longer needs to process the list, a call to QGYCLST should be done.

The open list APIs return data for use by the process open list APIs. The process open list APIs are located in the Process Open List category, whereas the open list APIs can be found in the applicable categories. For example, the Open List of Messages (QGYOLMSG) API is located in the Message Handling category.

The following table shows the format of the list information parameter in the open list APIs. For a detailed description of each field, see "Field descriptions."

Offset		Type	Field
Dec	**Hex**		
0	0	BINARY(4)	Total records
4	4	BINARY(4)	Records returned
8	8	CHAR(4)	Request handle
12	C	BINARY(4)	Record length
16	10	CHAR(1)	Information complete indicator
17	11	CHAR(13)	Date and time created
30	1E	CHAR(1)	List status indicator
31	1F	CHAR(1)	Reserved
32	20	BINARY(4)	Length of information returned
36	24	BINARY(4)	First record in receiver variable
40	28	CHAR(40)	Reserved

Field descriptions

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

Character	Description
1	Century, where 0 indicates years 19 *xx* and 1 indicates years 20 *xx*.
2-7	The date, in YYMMDD (year, month, day) format.
8-13	The time of day, in HHMMSS (hours, minutes, seconds) format.

First record in receiver variable. The number of the first record returned in the receiver variable.

Information complete indicator. Whether all requested information has been supplied. Possible values follow.

Value	Description
C	Complete and accurate information. All of the requested records have been returned in the receiver variable.
I	Incomplete information. An interruption causes the receiver variable to contain incomplete information.
P	Partial and accurate information. Partial information is returned when the receiver variable is full and not all of the records requested are returned.

Length of information returned. The size, in bytes, of the information that is returned in the receiver variable.

List status indicator. The status of building the list. Possible values follow.

Value	Description
0	The building of the list is pending.
1	The list is in the process of being built.
2	The list has been completely built.
3	An error occurred when building the list. The next call to the Get List Entries (QGYGTLE) API will cause the error to be signaled to the caller of the QGYGTLE API.
4	The list is primed and ready to be built. The list will be built asynchronously by a server job, but the server job has not necessarily started building the list yet.
5	Given the current selection criteria and information requested, there is too much data to be returned. The list is incomplete, but data collected to this point is available.

Record length. The length of each record of information returned. For variable length records, this value is set to zero. For variable length records, you can access the next record in the list by using **Offset** to the next entry, **Displacement** to the next entry, or **Length** of this entry, which is provided with each list entry returned.

Records returned. The number of records that are returned in the receiver variable. This number is the smallest of the following values:

- The number of records that will fit into the receiver variable.
- The number of records in the list.
- The number of records that are requested.

Request handle. The handle of the request that can be used for subsequent requests of information from the list. The handle is valid until the Close List (QGYCLST) API is called to close the list, or until the job ends.

Note: This field should be treated as a hexadecimal field. It should not be converted from one CCSID to another, for example, EBCDIC to ASCII, because doing so could result in an unusable value.

Reserved. An ignored field.

Total records. The total number of records available in the list.

Related reference:

Process Open List APIs

Path name format

The APIs that work with objects supported across file systems use a common path name format to identify the objects.

The format of the path name is as follows. For a detailed description of each field, see “Field descriptions” on page 119.

Offset		Use	Type	Field
Dec	Hex			
0	0	INPUT	BINARY(4)	CCSID
4	4	INPUT	CHAR(2)	Country or region ID
6	6	INPUT	CHAR(3)	Language ID
9	9	INPUT	CHAR(3)	Reserved
12	C	INPUT	BINARY(4)	Path type indicator
16	10	INPUT	BINARY(4)	Length of path name
20	14	INPUT	CHAR(2)	Path name delimiter character
22	16	INPUT	CHAR(10)	Reserved
32	20	INPUT	CHAR(*)	Path name

Field descriptions

This section describes the path name format fields in further detail. Field descriptions are in alphabetical order.

CCSID. The CCSID (coded character set ID) the path name is in. The possible values follow.

Value	Description
0	Use the current job default CCSID.
1-65533	A valid CCSID in this range.

Country or region ID. The country or region ID for the path name. The possible values follow.

Value	Description
X'0000'	Use the current job country or region ID.
Country or region ID	A valid country or region ID.

Language ID. The language ID for the path name. The possible values follow.

Value	Description
X'000000'	Use the current job language ID.
Language ID	A valid language ID.

Length of path name. The length of the path name in bytes.

Path name. Depending on the path type indicator field, this field contains either a pointer to a character string that contains the path name, or a character string that contains the path name.

The path name must be an absolute path name or a relative path name. An absolute path name is a path name that starts with the path name delimiter, usually the slash (/) character. A relative path name is a path name that does not start with the path name delimiter. When a relative name is specified, the API assumes that this path name starts at the current directory of the process that the API is running in.

The dot and dot dot (. ..) directories are valid in the path name. The home directory, generally represented by using the tilde character in the first character position of the path name, is not supported.

A null character value is not allowed as one of the characters in the path name unless a null character is specified as a path name delimiter.

To avoid confusion with IBM i special values, path names cannot start with a single asterisk (*) character.

Path name delimiter character. The delimiter character used between the element names in the path name. This is in the same CCSID as the path name. The most common delimiter is the slash (/) character. If the delimiter is 1 character, the first character of the 2-character field is used.

Path type indicator. Whether the path name contains a pointer or is a character string and whether the path name delimiter character is 1 or 2 characters long. The possible values follow.

Value	Description
0	The path name is a character string, and the path name delimiter character is 1 character long.
1	The path name is a pointer, and the path name delimiter character is 1 character long.
2	The path name is a character string, and the path name delimiter character is 2 characters long.
3	The path name is a pointer, and the path name delimiter character is 2 characters long.

Reserved. A reserved field that must be set to hexadecimal zeros.

Related information:

Integrated file system

Using APIs

These examples show how to use program-based APIs and service-program-based APIs.

Examples: Program-based APIs

These examples demonstrate the use of program-based APIs in several different high-level language programs.

The examples focus on descriptions, formats, variable-length fields as output, and optional parameters. They access information from a job description to demonstrate how to code APIs. While this might not be what your application requires, you can use the same approach to access information when you use most of the APIs.

Assume that you are interested in accessing the value of the HOLD parameter on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command. The HOLD parameter determines whether the job is held on the job queue. The following values are supported:

***NO**: The job is not held.

***YES**: The job is held on the job queue.

The first step is to find the correct API to use. To do this, you must identify the part of the IBM i licensed program that is most closely related to the function in which you are interested. If you want to access information from a job description, as in these examples, you need to know that a job description object is considered part of the work management function. API names contain verbs that are similar to the IBM i licensed program: change, create, remove, and retrieve. These examples use the Retrieve Job Description Information (QWDRJOBD) API.

For a detailed description of how to use the API, see “API information format” on page 47. These descriptions and the programs that support them are in RPG. You can, however, view the same programs in different languages.

Related concepts:

“APIs for the program-based environment” on page 10
Program-based APIs are called as programs (*PGMs). They are the initial APIs on the system.

Related reference:

“API naming conventions” on page 6
Program-based APIs and service-program-based APIs follow similar naming conventions.

Retrieve Job Description Information (QWDRJOBD) API

Example in OPM RPG: Retrieving the HOLD parameter (exception message)

This OPM RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

To make the RPG program JOBDAPI (this program name is also used in other examples in this topic collection) more general purpose, two parameters for the job description (JOBD) name and library (JOBDL) name are passed to it, as shown at (5). A message is sent for the value found. The program does not handle errors. Any errors are returned as exception messages.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 571.

```
I*****************************************************************
I*****************************************************************
I*
I*Program Name: JOBDAPI
I*
I*Language:  OPM RPG
I*
I*Descriptive Name:  Job Description
I*
I*Description:  This example expects errors to be sent as escape
I*              messages.
I*
I*Header Files Included: QUSEC - Error Code Parameter
I*                       QWDRJOBD - Retrieve Job Description API
I*
I*****************************************************************
I*****************************************************************
I*
I* Error Code Parameter Include for the APIs
I*
I/COPY QSYSINC/QRPGSRC,QUSEC
I*
I* Retrieve Job Description API Include
I*
I/COPY QSYSINC/QRPGSRC,QWDRJOBD          (1)
I*
I* Command String Data Structure
I*
ICMDSTR      DS
I I            'SNDMSG MSG(''HOLD -      1  26 CMD1
I              'value is '
I                                       27  36 HOLD
I I            ''') TOUSR(QPGMR)'       37  51 CMD2
I*
I* Miscellaneous Data Structure
I*
I            DS
I*                               (2)
```

```
I I            390                   B   1   40RCVLEN
I I            'JOBD0100'                5  12 FORMAT
I*               (3)
C*
C* Beginning of Mainline
C*
C* Two parameters are being passed into this program.
C*
C           *ENTRY    PLIST   (5)
C                     PARM           JOBD   10
C                     PARM           JOBDL  10
C*
C* Move the two parameters passed into LFNAM.
C*
C           JOBD      CAT  JOBDL     LFNAM  20   (6)
C* Error code bytes provided is set to 0
C*
C                     Z-ADD0         QUSBNB    (4)
C*
C* Instead of specifying 'QWCRJOBD', I could have used the
C* constant QWDBGB that was defined in the QWDRJOBD include.
C*
C                     CALL 'QWDRJOBD'
C                     PARM           QWDBH            Receiver Var.
C                     PARM           RCVLEN           Length RCVVAR
C                     PARM           FORMAT           Format Name
C                     PARM           LFNAM            Qual. Job Desc
C                     PARM           QUSBN            Error Code
C*
C                     MOVELQWDBHN    HOLD
C*
C* Let's tell everyone what the hold value was for this jobd.
C*
C                     Z-ADD51        LENSTR 155
C                     CALL 'QCMDEXC'
C                     PARM           CMDSTR
C                     PARM           LENSTR
C*
C                     SETON                     LR
C                     RETRN
C*
C* End of MAINLINE
C*
```

The program declares the variables to be used. The QWDBH variable is length 390 as shown at (2).

In the example, the program places a value of JOBD0100 in the format variable. A literal could have been used instead for those languages that support a literal on a call, as shown at (5). The program generates the qualified name of the job description (JOBD) by concatenating the simple name and the library qualifier, as shown at (6). A 20-character variable must be used, and the simple name must begin in byte 1 with the library qualifier in byte 11. Because CAT is used, a simple concatenation of two 10-byte variables occurs so that the names are in the correct place for the LFNAM parameter.

The QWDRJOBD API is called with the correct parameter list. The API uses the parameter list and accesses the job description specified. The API extracts the values from the internal object form and places them in a data structure that matches the JOBD0100 format. The API then returns with the data structure placed in variable QWDBH, which is located in member QWDRJOBD in the QSYSINC library.

The output is similar to the following:

```
                      Display Messages

                                                        System:   GENSYS90
Queue . . . . . :   QPGMR                  Program . . . . :   *DSPMSG
  Library . . . :     QUSRSYS                Library . . . :
Severity  . . . :   00                     Delivery  . . . :   *HOLD
Type reply (if required), press Enter.
  From  . . . :   SMITH          07/23/94   10:25:14
  HOLD value is *NO
```

The API does not need to be called each time that you want a separate field because all fields are returned that would fit within the size indicated by the length of receiver variable (RCVLEN) parameter. You can run the program against the QBATCH job description in library QGPL by using the following call statement:

```
CALL JOBDAPI PARM(QBATCH QGPL)
```

If QGPL is on the library list, you can run the program against the QBATCH job description by using the following call statement:

```
CALL JOBDAPI PARM(QBATCH *LIBL)
```

You can run the program on one of your own job descriptions or on a test job description where you have specified HOLD(*YES).

Example in OPM RPG: Handling error conditions

For this example, assume that the XYZ job description does not exist:

```
CALL JOBDAPI PARM(XYZ *LIBL)
```

You probably will receive the inquiry message CPA0701 that states an unmonitored exception (CPF9801) has occurred and offers several possible replies. At this point, you would enter C for Cancel and press the Enter key.

If you displayed the low-level messages, you would see the following: CPF9801 (Object not found), followed by the inquiry message (CPA0701), followed by your reply.

When you specify the error code parameter as zero, you are specifying that exceptions be sent as escape messages. You can code the RPG program so that any errors on the call set the indicator 01 to on, as shown at (10). This causes a different path to be taken in the code.

For RPG, the CALL operation specifies the error indicator. Based on whether the error indicator is on or off, a set of instructions can be processed. The API must receive an error code parameter that consists of a binary 4 field with a value of binary zeros, as shown at (11)). The message ID can be accessed from the program-status data structure. You would define this as follows:

```
     I* Program status DS ((12))
     IPGMSTS     SDS
     I                                       40  46 MSGIDD
```

If you are going to do something about an error condition, you must test for an error condition in RPG:

- If you use the error-code data structure, test the bytes available field.
- If you let exceptions occur, test the error indicator on the CALL operation ((10)).

Because you must test for some condition (one of the error messages in Error Messages), no great difference exists in how you handle error conditions in RPG. The error-code data structure is a little more straightforward (the program-status data structure is not used). The only disadvantage of the error-code data structure is that the escape message that occurred was removed from the job log.

The following program shows how to code for an error condition, test for that condition, and send a message to the QPGMR message queue if the condition occurs:

```
I*****************************************************************
I*****************************************************************
I*
I*Program Name: JOBDAPI
I*
I*Language:  OPM RPG
I*
I*Descriptive Name:  Get Job Description
I*
I*Description:  This program handles any errors that are
I*              returned
I*
I*Header Files Included: QUSEC - Error Code Parameter
I*                       QWDRJOBD - Retrieve Job Description API
I*
I*****************************************************************
I*****************************************************************
I*
I* Error Code Parameter Include for the APIs
I*
I/COPY QSYSINC/QRPGSRC,QUSEC
I*
I* Retrieve Job Description API Include
I*
I/COPY QSYSINC/QRPGSRC,QWDRJOBD
I* Program status DS
IPGMSTS     SDS   (12)
I                                       40  46 MSGIDD
I*
I* Command String Data Structure
I*
ICMDSTR      DS
I I            'SNDMSG MSG(''HOLD -      1  26 CMD1
I              'value is '
I                                       27  36 HOLD
I I            ''') TOUSR(QPGMR)'       37  51 CMD2
I*
IMSG3        DS
I I            'SNDMSG MSG(''No such-    1  35 MSG3A
I              ' *JOBD exists'') '
I I            'TOUSR(QPGMR)'           36  47 MSG3B
I*
I* Miscellaneous Data Structure
I*
I            DS
I I            390                   B   1   40RCVLEN
I I            'JOBD0100'                5  12 FORMAT
C*
C* Beginning of Mainline
C*
C* Two parameters are being passed into this program.
C*
C           *ENTRY    PLIST
C                     PARM           JOBD   10
C                     PARM           JOBDL  10
C*
C* Move the two parameters passed into LFNAM.
C*
C           JOBD      CAT  JOBDL     LFNAM  20
C* Error code bytes provided is set to 0
C*
C                     Z-ADD0         QUSBNB (11)
C*
C* Instead of specifying 'QWCRJOBD', I could have used the
```

```
C* constant QWDBGB that was defined in the QWDRJOBD include.
C*
C                     CALL 'QWDRJOBD'             01 (10)
C                     PARM           QWDBH            Receiver Var.
C                     PARM           RCVLEN           Length RCVVAR
C                     PARM           FORMAT           Format Name
C                     PARM           LFNAM            Qual. Job Desc
C                     PARM           QUSBN            Error Code
C   01                EXSR ERROR                      Error Subroutine
C*
C  N01                MOVELQWDBHN    HOLD
C*
C* Let's tell everyone what the hold value was for this job.
C*
C  N01                Z-ADD51        LENSTR 155
C  N01                CALL 'QCMDEXC'
C                     PARM           CMDSTR
C                     PARM           LENSTR
C*
C                     SETON                      LR
C                     RETRN
C*
C* End of MAINLINE
C*
C* Subroutine to handle errors received on the CALL
C*
C           ERROR     BEGSR
C           MSGIDD    IFEQ 'CPF9801'
C*
C* Process errors returned from the API.
C*
C                     Z-ADD47        LENSTR 155
C                     CALL 'QCMDEXC'
C                     PARM           MSG3
C                     PARM           LENSTR
C                     END
C                     ENDSR
```

If the CPF9801 exception occurs, your program sends a message to the QPGMR message queue as shown in the following display:

```
                      Display Messages

                                                    System:   GENSYS90
Queue . . . . . :   QPGMR                 Program . . . . :   *DSPMSG
  Library . . . :     QUSRSYS               Library . . . :
Severity  . . . :   00                    Delivery  . . . :   *HOLD
Library . . . :
Type reply (if required), press Enter.
  From  . . . :   SMITH         07/25/94   11:10:12
  No such *JOBD exists
```

If another exception occurs (for example, a library name that is not valid), you do not receive an indication that an error occurred because of the way the error subroutine is currently coded.

In addition, you can use the Message Handling APIs to receive the messages sent to your program message queue.

The call to the API fails if you specify a valid job description but use a library qualifier such as *ALLUSR. The value *ALLUSR is not supported by the description of the required parameter group.

Related reference:

Retrieve Job Description Information (QWDRJOBD) API

"Example in ILE COBOL: Retrieving the HOLD parameter (exception message)"
This ILE COBOL program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

"Example in ILE C: Retrieving the HOLD parameter (exception message)" on page 127
This ILE C program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

"Example in ILE RPG: Retrieving the HOLD parameter (exception message)" on page 129
This ILE RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

Example in ILE COBOL: Retrieving the HOLD parameter (exception message)

This ILE COBOL program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

The following program also works for OPM COBOL.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*Program Name:          JOBDAPI
*
*Programming Language:  COBOL
*
*Description:           This example expects errors sent as
*                       escape messages.
*
*Header Files Included: QUSEC - Error Code Parameter
*                       QWDRJOBD - Retrieve Job Description API
*
*****************************************************************
*****************************************************************
*
 PROGRAM-ID. JOBDAPI.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Retrieve Job Description API Include
*
 COPY QWDRJOBD OF QSYSINC-QLBLSRC.  (2)
*
* Command String Data Structure
*
 01  COMMAND-STRING.
     05  TEXT1 PIC X(26) VALUE 'SNDMSG MSG(''HOLD value is'.
     05  HOLD  PIC X(10).
     05  TEXT2 PIC X(15) VALUE ''') TOUSR(QPGMR)'.
*
 01  COMMAND-LENGTH PIC S9(10)V99999 COMP-3.
 01  RECEIVER-LENGTH PIC S9(9) COMP-4.  (4)
 01  FORMAT-NAME PIC X(8) VALUE 'JOBD0100'.  (5)
 01  QCMDEXC PIC X(10) VALUE 'QCMDEXC'.
*
* Job Description and Library Name Structure
```

```
      *
       01  JOBD-AND-LIB-NAME.
           05  JOB-DESC PIC X(10).
           05  JOB-DESC-LIB PIC X(10).
      *
       LINKAGE SECTION.
      *
      * Two Parameters are being passed into this program.
      *
       01  JOBD PIC X(10).
       01  JOBDL PIC X(10).
      *
       PROCEDURE DIVISION USING JOBD, JOBDL.  (8)
       MAIN-LINE.
      *
      * Beginning of Mainline
      *
      * Move the two parameters passed into JOB-DESC and JOB-DESC-LIB.  (9)
        MOVE JOBD TO JOB-DESC.
           MOVE JOBDL TO JOB-DESC-LIB.
      *
      * Error Code Parameter is set to 0.
      *
           MOVE 0 TO BYTES-PROVIDED.      (6)
      *
      * Receiver Length Set to 390.
      *
           MOVE 390 TO RECEIVER-LENGTH.     (3)
      *
      * Call the QWDRJOBD API.
      *
           CALL QWDRJOBD USING QWD-JOBD0100, RECEIVER-LENGTH,
                   FORMAT-NAME, JOBD-AND-LIB-NAME, QUS-EC.
      *
      * Move HOLD-JOB-QUEUE to HOLD so that we can display the value using
      * the command string.
      *
           MOVE HOLD-JOB-QUEUE TO HOLD.
      *
      * Let's tell everyone what the hold value was for this job.
      *
           MOVE 51 TO COMMAND-LENGTH.
           CALL QCMDEXC USING COMMAND-STRING, COMMAND-LENGTH.
      *
           STOP RUN.
```

Related reference:

"Example in OPM RPG: Retrieving the HOLD parameter (exception message)" on page 121
This OPM RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

Example in ILE C: Retrieving the HOLD parameter (exception message)

This ILE C program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*Program Name:          JOBDAPI                                     */
/*                                                                   */
/*Programming Language:  ILE C                                       */
```

```
/*                                                                    */
/*Description:           This example expects errors sent as          */
/*                       escape messages.                             */
/*                                                                    */
/*Header Files Included: SIGNAL - C Error Signalling Routines         */
/*                       STDIO - Standard Input/Output                */
/*                       STRING - String Functions                    */
/*                       QUSEC - Error Code Parameter                 */
/*                       QWDRJOBD - Retrieve Job Description API      */
/*                       QLIEPT - Entry Point Table                   */
/*                                                                    */
/**********************************************************************/
/**********************************************************************/

#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <qusec.h>        /* Error Code Parameter Include for the APIs */
#include <qwdrjobd.h>  (2)
                          /* Retrieve Job Description API Include      */
#include <qliept.h>

char received[8];
                          /* Used to receive error msgs signaled       */
                          /* from QWDRJOBD API.                        */

/**********************************************************************/
/* Function:     error_handler                                        */
/* Description:  This function handles exceptions signalled from the  */
/*               QWDRJOBD API.  The message identifier received is    */
/*               assigned to the variable 'received'.                 */
/**********************************************************************/

void error_handler(int dummy)
{
  _INTRPT_Hndlr_Parms_T ExcDta = {0};

  _GetExcData(&ExcDta);
  memcpy(received,ExcDta.Msg_Id,7);
  signal(SIGALL,error_handler);
}

/**********************************************************************/
/* Error Code Structure                                               */
/*                                                                    */
/* This shows how the user can define the variable length portion of  */
/* error code for the exception data.                                 */
/*                                                                    */
/**********************************************************************/
typedef struct {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
        } error_code_t;

main(int argc, char *argv[]  (8)
{
  error_code_t  error_code;
  char          qual_job_desc[20];
  char          *qual_job_ptr = qual_job_desc;
  char          rec_var[390];
  char          hold_value[10];
  char          command_string[53];

  /********************************************************************/
  /* Enable error handler.                                            */
  /********************************************************************/
  signal(SIGALL,error_handler);
```

```
  memset(hold_value, ' ', 10);
  memset(received, ' ', 7);

  /*********************************************************************/
  /* Make sure we received the correct number of parameters. The argc  */
  /* parameter will contain the number of parameters that was passed   */
  /* to this program. This number also includes the program itself,    */
  /* so we need to evaluate argc-1.                                    */
  /*********************************************************************/

  if (((argc - 1) < 2) || ((argc - 1 > 2)))
  /*********************************************************************/
  /* We did not receive all of the required parameters so exit the     */
  /* program.                                                          */
  /*********************************************************************/
  {
    exit(1);
  }

  /*********************************************************************/
  /* Move the two parameters passed into qual_job_desc.  (9)           */
  /*********************************************************************/
  memcpy(qual_job_ptr, argv[1], 10);
  qual_job_ptr += 10;
  memcpy(qual_job_ptr, argv[2], 10);  (6)

  /*********************************************************************/
  /* Set the error code parameter to 0.                                */
  /*********************************************************************/
  error_code.ec_fields.Bytes_Provided = 0;

  /*********************************************************************/
  /* Call the QWDRJOBD API.                                            */
  /*********************************************************************/
  QWDRJOBD(rec_var,               /* Receiver Variable                 */
           390,  (3)         /* Receiver Length                        */
           "JOBD0100",  (5) /* Format Name                             */
           qual_job_desc,         /* Qualified Job Description         */
           &error_code);          /* Error Code                        */

  if(memcmp(received, "       ", 7) == 0)
    memcpy(hold_value, ((Qwd_JOBD0100_t *)rec_var)->Hold_Job_Queue, 10);

  /*********************************************************************/
  /* Let's tell everyone what the hold value was for this job.         */
  /*********************************************************************/
  sprintf(command_string,
          "SNDMSG MSG('HOLD value is %.7s') TOUSR(QPGMR)",
          hold_value);
  system(command_string);

} /* main */
```

Related reference:

"Example in OPM RPG: Retrieving the HOLD parameter (exception message)" on page 121
This OPM RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

Example in ILE RPG: Retrieving the HOLD parameter (exception message)

This ILE RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
D*************************************************************
D*************************************************************
D*
D*  Program Name: JOBDAPI
D*
D*  Programming Language: ILE RPG
D*
D*  Description:  This program retrieves the HOLD value from
D*                a job description.  It expects errors to be
D*                sent as escape messages.
D*
D*  Header Files Included: QUSEC - Error Code Parameter
D*                         QWDRJOBD - Retrieve Job Description API
D*
D*************************************************************
D*************************************************************
D*
D* Error Code parameter include
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D* Retrieve Job Description API Include
D*
D/COPY QSYSINC/QRPGLESRC,QWDRJOBD      (2)
D*
D* Command string data structure
D*
DCMD_STRING       DS
D                                26    INZ('SNDMSG MSG(''HOLD value is ')
D HOLD                           10
D                                15    INZ(''') TOUSR(QPGMR)')
D*
D* Miscellaneous data structure
D*                               (4)       (2)  (3)
DRCVLEN           S               9B 0 INZ(%SIZE(QWDD0100))
DFORMAT           S               8    INZ('JOBD0100')  (5)
DLENSTR           S              15  5 INZ(%SIZE(CMD_STRING))
C*
C* Beginning of mainline
C*
C* Two parameters are being passed into this program
C*
C     *ENTRY        PLIST  (8)
C                   PARM                    JOBD            10
C                   PARM                    JOBD_LIB        10
C*
C* Move the two parameters passed into LFNAM
C*
C     JOBD          CAT       JOBD_LIB      LFNAM           20  (9)
C*
C* Error Code Bytes Provided is set to 0
C*
C                   Z-ADD     0             QUSBPRV    (6)
C*
C* Call the API.
C*
C                   CALL      QWDRJOBD
C                   PARM                    QWDD0100
C                   PARM                    RCVLEN
C                   PARM                    FORMAT
C                   PARM                    LFNAM
C                   PARM                    QUSEC
C*
C                   MOVEL     QWDHJQ        HOLD
```

```
C*
C* Let's tell everyone what the hold value was for this job
C*
C                   CALL      'QCMDEXC'
C                   PARM                    CMD_STRING
C                   PARM                    LENSTR
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C*
C* End of MAINLINE
C*
```

Example in ILE RPG: Handling error conditions

This program can be written only in OPM RPG and ILE RPG.

```
D*****************************************************************
D*****************************************************************
D*
D*  Program Name: JOBDAPI
D*
D*  Programming Language: ILE RPG
D*
D*  Description:  This program retrieves the HOLD value from
D*                a job description.  It expects errors to be
D*                sent as escape messages.
D*
D*  Header Files Included: QUSEC - Error Code Parameter
D*                         QWDRJOBD - Retrieve Job Description API
D*
D*****************************************************************
D*****************************************************************
D*
D* Error Code parameter include
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D* Retrieve Job Description API Include
D*
D/COPY QSYSINC/QRPGLESRC,QWDRJOBD
D*
D* Program status DS
D*
DPGMSTS          SDS   (12)
D MSG_ID                 40     46
D*
D* Command string data structure
D*
DCMD_STRING      DS
D                                26    INZ('SNDMSG MSG(''HOLD value is ')
D HOLD                           10
D                                15    INZ(''') TOUSR(QPGMR)')
D*
D* Miscellaneous data structure
D*
DRCVLEN          S                9B 0 INZ(%SIZE(QWDD0100))
DFORMAT          S                8    INZ('JOBD0100')
DLENSTR          S               15  5 INZ(%SIZE(CMD_STRING))
DNO_JOBD         S               47    INZ('SNDMSG MSG(''No such *JOBD -
D                                      exists'') TOUSR(QPGMR)')
DNO_JOBD_SZ      S               15  5 INZ(%SIZE(NO_JOBD))
C*
C* Beginning of mainline
C*
C* Two parameters are being passed into this program
C*
```

```
C     *ENTRY        PLIST
C                   PARM                    JOBD            10
C                   PARM                    JOBD_LIB        10
C*
C* Move the two parameters passed into LFNAM
C*
C     JOBD          CAT       JOBD_LIB      LFNAM           20
C*
C* Error Code Bytes Provided is set to 0
C*
C                   Z-ADD     0             QUSBPRV    (11)
C*
C* Call the API.
C*
C                   CALL      QWDRJOBD                           01  (10)
C                   PARM                    QWDD0100
C                   PARM                    RCVLEN
C                   PARM                    FORMAT
C                   PARM                    LFNAM
C                   PARM                    QUSEC
C*
C* Test for an error on the API call
C*
C                   IF        *IN01 = *ON
C*
C* If there was an error, exit to ERROR subroutine
C*
C                   EXSR      ERROR
C*
C* Else, process the HOLD value
C*
C                   ELSE
C                   MOVEL     QWDHJQ        HOLD
C*
C* Let's tell everyone what the hold value was for this job
C*
C                   CALL      'QCMDEXC'
C                   PARM                    CMD_STRING
C                   PARM                    LENSTR
C                   END
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C*
C* End of MAINLINE
C*
C* Subroutine to handle errors received on the CALL
C*
C     ERROR         BEGSR
C                   IF        MSG_ID = 'CPF9801'
C*
C* Process errors returned from the API
C*
C                   CALL      'QCMDEXC'
C                   PARM                    NO_JOBD
C                   PARM                    NO_JOBD_SZ
C                   END
C                   ENDSR
```

Related reference:

"Example in OPM RPG: Retrieving the HOLD parameter (exception message)" on page 121
This OPM RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned as exception messages.

Example in OPM RPG: Retrieving the HOLD parameter (error code structure)

This OPM RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

In "Example in OPM RPG: Retrieving the HOLD parameter (exception message)" on page 121, QUSBNB was set to a value of binary zero to tell the API to send exceptions (escape messages) for any error conditions. The example in this topic uses an error-code data structure as an alternative to receiving exceptions.

Some languages do not support the use of exceptions, so you might prefer to code for errors using error code structures.

In your programs, you can use error code structures in the following ways:

- Define an 8-byte error code structure that provides feedback on whether an error occurred. If an error does occur, you are not able to determine the specifics of the problem.
- Define a 16-byte error code structure that allows you to determine if an error exists and to access the exception message ID. The exception message IDs are the same as shown in Error messages.
- Define a larger than 16-byte error code structure that provides the same information as described in the previous two error code structures as well as some or all of the exception data. The exception data is the message data that is sent with the exception message. Because the vast majority of exception messages do not have more than 512 bytes of message data, a 600-byte error code structure would be adequate for almost all cases.

Note: Lengths of 1 through 7 bytes are not valid for the error code structure.

Format of an error code structure

The format of the error code structure (QUSBN) follows.

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

The error code structure can be found in the QUSEC member in the QSYSINC library, as shown at (1). Which of the files you use depends on the language.

The bytes provided field describes the size of the error code structure that you declared in your program and how you want errors returned. (This was set to 0 as shown by **(6)** in Example in OPM RPG: Retrieving the HOLD parameter (exception message).)

The bytes available field describes how many bytes the API could have passed back. If this field is zero, no exception occurred. The correct method for testing if an error occurred when using a nonzero-bytes-provided value is to check this field for a value greater than zero, as shown at (2).

The exception ID is the normal 7-character message ID, such as CPF9801, that occurs for an object-not-found condition. Do not test this field to determine if an error exists. The field is properly set

by the system only if the number of bytes available is greater than 0. Similarly, the exception data (message data) information is not set properly unless an error exists; for example, any information left from a prior call is not changed.

The following program is the same as the previous program except that a 16-byte error code structure is used:

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
I*****************************************************************
I*****************************************************************
I*
I*Program Name: JOBDAPI
I*
I*Language:  OPM RPG
I*
I*Descriptive Name:  Get Job Description
I*
I*Description: This sample program shows exceptions being
I*             returned in the error code parameter.
I*
I*Header Files Included: QUSEC - Error Code Parameter
I*                       QWDRJOBD - Retrieve Job Description API
I*
I*****************************************************************
I*****************************************************************
I*
I* Error Code Parameter Include for the APIs
I*
I/COPY QSYSINC/QRPGSRC,QUSEC                       (1)
I*
I* Retrieve Job Description API Include
I*
I/COPY QSYSINC/QRPGSRC,QWDRJOBD
I*
I* Command String Data Structure
I*
ICMDSTR      DS
I I            'SNDMSG MSG(''HOLD -      1  26 CMD1
I              'value is '
I                                       27  36 HOLD
I I            ''') TOUSR(QPGMR)'       37  51 CMD2
I*
IMSG2        DS
I I            'SNDMSG MSG(''Progr-      1  43 MSG2A
I              'am failed with mes-
I              'sage ID '
I                                       44  50 MSGIDD
I I            ''') TOUSR(QPGMR)'       51  65 MSG2B
I*
I* Miscellaneous Data Structure
I*
I            DS
I I            390                  B   1   40RCVLEN
I I            'JOBD0100'               5  12 FORMAT
C*
C* Beginning of Mainline
C*
C* Two parameters are being passed into this program.
C*
C           *ENTRY    PLIST
C                     PARM           JOBD   10
C                     PARM           JOBDL  10
C*
C* Move the two parameters passed into the LFNAM.
```

```
C*
C           JOBD      CAT  JOBDL     LFNAM  20
C*
C* Error code parameter is set to 16
C*
C                     Z-ADD16        QUSBNB                  (3)
C*
C* Instead of specifying 'QWCRJOBD', I could have used the
C* constant QWDBGB that was defined in the QWDRJOBD include.
C*
C                     CALL 'QWDRJOBD'
C                     PARM           QWDBH            Receiver Var.
C                     PARM           RCVLEN           Length RCVVAR
C                     PARM           FORMAT           Format Name
C                     PARM           LFNAM            Qual. Job Desc
C                     PARM           QUSBN            Error Code
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C*
C*
C  N01                MOVELQWDBHN    HOLD
C*
C* Let's tell everyone what the hold value was for this job.
C*
C  N01                Z-ADD51        LENSTR 155
C  N01                CALL 'QCMDEXC'
C                     PARM           CMDSTR
C                     PARM           LENSTR
C*
C                     SETON                     LR
C                     RETRN
C*
C* End of MAINLINE
C*
C*
C* Subroutine to handle errors returned in the error code
C* parameter.
C*
C           ERRCOD    BEGSR
C           QUSBNC    IFGT 0                     (2)
C*
C* Process errors returned from the API.
C*
C                     SETON                     01
C                     Z-ADD65        LENSTR 155
C                     MOVELQUSBND    MSGIDD
C                     CALL 'QCMDEXC'
C                     PARM           MSG2
C                     PARM           LENSTR
C                     END
C                     ENDSR
```

The QUSBN error-code data structure is defined in the QUSEC include file, as shown at (1). The program initializes the bytes provided field (QUSBNB) with a value of 16, as shown at (3). This sets the first field of the error code structure to tell the API not to send an exception but to use the first 16 bytes of the QUSBN parameter to return the error information. After the call to the API, the program accesses the bytes available (QUSBNC), as shown at (2). This contains the number of bytes of information about the error condition. The program is coded such that it tests if the number exceeds zero. This is the correct method of determining whether an error has occurred.

If an error occurred, you might want to handle the error in many different methods. The program shown extracts the specific error message ID that occurred and sends the 7-character value as a message. The

QUSBN parameter is used for both input and output (see Format of an error code structure). The first 4 bytes are input to the API to tell it how to handle exceptions. The remaining bytes are output from the API about any exception conditions.

To see the value of the HOLD attribute, use the following call statement to run the program against the QBATCH job description in library QGPL:

```
CALL JOBDAPI (QBATCH QGPL)
```

You should see that the value of the HOLD attribute is *NO:

```
+------------------------------------------------------------------------------+
|                                                                              |
|                                                                              |
|                             Display Messages                                 |
|                                                                              |
|                                                                              |
|                                                  System:   GENSYS90          |
| Queue . . . . . :   QPGMR                  Program . . . . :   *DSPMSG       |
|   Library . . . :     QUSRSYS                Library . . . :                   |
| Severity  . . . :   00                     Delivery  . . . :   *HOLD         |
| Type reply (if required), press Enter.                                       |
|   From  . . . :   SMITH          07/23/94   10:25:14                         |
|   HOLD value is *NO                                                          |
|                                                                              |
+------------------------------------------------------------------------------+
```

Example in OPM RPG: Handling error conditions

For this error condition, you should assume that the XYZ job description does not exist. Use the following call statement to run the error condition:

```
CALL JOBDAPI (XYZ *LIBL)
```

You should see that the CPF9801 message (Object not found) was issued:

```
+------------------------------------------------------------------------------+
|                                                                              |
|                                                                              |
|                             Display Messages                                 |
|                                                                              |
|                                                  System:   GENSYS90          |
| Queue . . . . . :   QPGMR                  Program . . . . :   *DSPMSG       |
|   Library . . . :     QUSRSYS                Library . . . :                   |
| Severity  . . . :   00                     Delivery  . . . :   *HOLD         |
| Type reply (if required), press Enter.                                       |
|   From  . . . :   SMITH          07/23/94   10:56:13                         |
|   Program failed with message ID CPF9801                                     |
|                                                                              |
|                                                                              |
+------------------------------------------------------------------------------+
```

Then run another error condition. For this error condition, you should assume that the XYZ library does not exist. Use the following call statement:

```
CALL JOBDAPI (QPGMR XYZ)
```

The output is similar to the following:

```
+------------------------------------------------------------------------------+
|                                                                              |
|                                                                              |
|                             Display Messages                                 |
|                                                                              |
|                                                  System:   GENSYS90          |
| Queue . . . . . :   QPGMR                  Program . . . . :   *DSPMSG       |
|   Library . . . :     QUSRSYS                Library . . . :                   |
```

```
| Severity  . . . :   00                     Delivery  . . . :   *HOLD          |
| Type reply (if required), press Enter.                                        |
|   From  . . . :   SMITH          07/23/94   10:56:13                          |
|   Program failed with message ID CPF9810                                      |
|                                                                               |
+-------------------------------------------------------------------------------+
```

You should see that the CPF9810 message (Library not found) was issued. An advantage of the error return variable is that it can contain other information such as message data. The following are the changes needed to return a 200-byte error code structure:

```
     I*****************************************************************
     I*****************************************************************
     I*
     I*Program Name: JOBDAPI
     I*
     I*Language:  OPM RPG
     I*
     I*Descriptive Name:  Get Job Description
     I*
     I*Description: This sample program shows the incorrect
     I*          way of using the offset in a user space in RPG.
     I*
     I*Header Files Included: QUSEC - Error Code Parameter
     I*                               (Copied into Program)
     I*                       QWDRJOBD - Retrieve Job Description API
     I*
     I*****************************************************************
     I* Error Code Parameter Include for the APIs
     I*
     I* The following QUSEC include is copied into this program
     I* so that the variable-length field can be defined as
     I* fixed length.
     I*
     I*** START HEADER FILE SPECIFICATIONS ****************************
     I*
     I*Header File Name: H/QUSEC
     I*
     I*Descriptive Name: Error Code Parameter.
     I*
     I*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
     I*All rights reserved.
     I*US Government Users Restricted Rights -
     I*Use, duplication or disclosure restricted
     I*by GSA ADP Schedule Contract with IBM Corp.
     I*
     I*Licensed Materials-Property of IBM
     I*
     I*
     I*Description: Include header file for the error code parameter.
     I*
     I*Header Files Included: None.
     I*
     I*Macros List: None.
     I*
     I*Structure List: Qus_EC_t
     I*
     I*Function Prototype List: None.
     I*
     I*Change Activity:
     I*
     I*CFD List:
     I*
     I*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
     I*---- ------------ ----- ------ --------- ----------------------
     I*$A0= D2862000     3D10  931201 DPOHLSON: New Include
     I*
```

```
I*End CFD List.
I*
I*Additional notes about the Change Activity
I*End Change Activity.
I*** END HEADER FILE SPECIFICATIONS *****************************
I*****************************************************************
I*Record structure for Error Code Parameter
I****                                                          ***
I*NOTE: The following type definition defines only the fixed
I*   portion of the format.  Varying-length field exception
I*   data is not defined here.
I*****************************************************************
IQUSBN       DS
I*                                          Qus EC
I                                    B   1   40QUSBNB
I*                                          Bytes Provided
I                                    B   5   80QUSBNC
I*                                          Bytes Available
I                                        9  15 QUSBND
I*                                          Exception Id
I                                       16  16 QUSBNF
I*                                          Reserved
I*                                       17  17 QUSBNG
I*
I*                                       Varying length
I                                        17 200 QUSBNG   (4)

.
.
.

C                    Z-ADD200       QUSBNB
C*
C                    CALL 'QWDRJOBD'
C                    PARM           QWDBH            Receiver Var.
C                    PARM           RCVLEN           Length RCVVAR
C                    PARM           FORMAT           Format Name
C                    PARM           LFNAM            Qual. Job Desc
C                    PARM           QUSBN            Error Code
```

The value placed in the QUSBNG variable (4) is the message data associated with the message ID that is identified as the exception. The message data follows the same format as if you had entered a Receive Message (RCVMSG) command and requested the message data (MSGDTA) parameter. You can use the Display Message Description (DSPMSGD) command to determine the layout of the message data for a particular message ID. When you handle exceptions, the only information provided is the exception ID and the message data associated with the exception. You cannot receive a diagnostic message (if one were sent in addition to the escape message) in the error-code data structure. You can use the message handling APIs to receive messages from your program message queue and to access the other messages that might be issued from the API.

When you instruct the API to return all errors in the error-code data structure, the escape message does not appear in the job log. The escape message not appearing in the job log is one of the major differences between letting the API return errors in an error-code data structure and letting the API send escape messages. For the error-code data structure, the escape messages have been removed from the job log by the API. If a diagnostic message is sent first, the diagnostic message exists in the job log and can be received.

Related concepts:

"Data types and APIs" on page 66
APIs support character data and binary data.

"Include files and the QSYSINC library" on page 59
An *Include file* is a text file that contains declarations that are used by a group of functions, programs, or users. The system include (QSYSINC) library provides all source include files for APIs that are included

with the IBM i operating system.

Related reference:

Retrieve Job Description Information (QWDRJOBD) API

"Example in ILE COBOL: Retrieving the HOLD parameter (error code structure)"
This ILE COBOL program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

"Example in ILE C: Retrieving the HOLD parameter (error code structure)" on page 141
This ILE C program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

"Example in ILE RPG: Retrieving the HOLD parameter (error code structure)" on page 143
This ILE RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

Example in ILE COBOL: Retrieving the HOLD parameter (error code structure)

This ILE COBOL program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

The following program also works for OPM COBOL.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 IDENTIFICATION DIVISION.
****************************************************************
****************************************************************
*
*Program Name:          JOBDAPI
*
*Programming Language:  COBOL
*
*Description:           This example shows how to make use of an
*                       error returned in the error code
*                       structure.
*
*Header Files Included: QUSEC - Error Code Parameter
*                       QWDRJOBD - Retrieve Job Description API
*
****************************************************************
****************************************************************
*
 PROGRAM-ID. JOBDAPI.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.                (1)
*
* Retrieve Job Description API Include
*
```

```
 COPY QWDRJOBD OF QSYSINC-QLBLSRC.
*
* Command String Data Structure
*
 01  COMMAND-STRING.
     05  TEXT1 PIC X(26) VALUE 'SNDMSG MSG(''HOLD value is'.
     05  HOLD  PIC X(10).
     05  TEXT2 PIC X(15) VALUE ''') TOUSR(QPGMR)'.
*
* Message Identifier Data Structure
*
 01  MESSAGE-TWO.
     05  MSG2A  PIC X(43)
         VALUE 'SNDMSG MSG(''Program failed with message ID'.
     05  MSGIDD PIC X(7).
     05  MSG2B  PIC X(15) VALUE ''') TOUSR(QPGMR)'.
*
 01  COMMAND-LENGTH PIC S9(10)V99999 COMP-3.
 01  RECEIVER-LENGTH PIC S9(9) COMP-4.
 01  FORMAT-NAME PIC X(8) VALUE 'JOBD0100'.
 01  QCMDEXC PIC X(10) VALUE 'QCMDEXC'.
*
* Job Description and Library Name Structure
*
 01  JOBD-AND-LIB-NAME.
     05  JOB-DESC PIC X(10).
     05  JOB-DESC-LIB PIC X(10).
*
 LINKAGE SECTION.
*
* Two Parameters are being passed into this program.
*
 01  JOBD PIC X(10).
 01  JOBDL PIC X(10).
*
 PROCEDURE DIVISION USING JOBD, JOBDL.
 MAIN-LINE.
*
* Beginning of Mainline
*
* Move the two parameters passed into JOB-DESC and JOB-DESC-LIB.
*
     MOVE JOBD TO JOB-DESC.
     MOVE JOBDL TO JOB-DESC-LIB.
*
* Error Code Parameter is set to 16.
*
     MOVE 16 TO BYTES-PROVIDED.             (3)
*
* Receiver Length Set to 390.
*
     MOVE 390 TO RECEIVER-LENGTH.
*
* Call the QWDRJOBD API.
*
     CALL QWDRJOBD USING QWD-JOBD0100, RECEIVER-LENGTH,
             FORMAT-NAME, JOBD-AND-LIB-NAME, QUS-EC.
*
* See if any errors were returned in the error code parameter.
*
     PERFORM ERRCOD.
*
* Move HOLD-JOB-QUEUE to HOLD so that we can display the value using
* the command string.
*
     MOVE HOLD-JOB-QUEUE TO HOLD.
*
```

```
      * Let's tell everyone what the hold value was for this job.
      *
           MOVE 51 TO COMMAND-LENGTH.
           CALL QCMDEXC USING COMMAND-STRING, COMMAND-LENGTH.
      *
           STOP RUN.
      *
      * End of Mainline
      *
      *
      * Subroutine to handle errors returned in the error code
      * parameter.
      *
       ERRCOD.
      *
           IF BYTES-AVAILABLE OF QUS-EC > 0          (2)
      *
      * Process errors returned from the API.
      *
             MOVE 65 TO COMMAND-LENGTH,
             MOVE EXCEPTION-ID TO MSGIDD,
             CALL QCMDEXC USING MESSAGE-TWO, COMMAND-LENGTH,
             STOP RUN.
```

Related reference:

"Example in OPM RPG: Retrieving the HOLD parameter (error code structure)" on page 133
This OPM RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

Example in ILE C: Retrieving the HOLD parameter (error code structure)

This ILE C program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*Program Name:          JOBDAPI                                     */
/*                                                                   */
/*Programming Language:  ILE C                                       */
/*                                                                   */
/*Description:           This example shows how to make use of an    */
/*                       error returned in the error code structure. */
/*                                                                   */
/*Header Files Included: STDIO - Standard Input/Output               */
/*                       STRING - String Functions                   */
/*                       QUSEC - Error Code Parameter                */
/*                       QWDRJOBD - Retrieve Job Description API     */
/*                       QLIEPT - Entry Point Table                  */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <stdio.h>
#include <string.h>
#include <qusec.h>     (1)   /* Error Code Parameter Include for the API  */
#include <qwdrjobd.h>     /* Retrieve Job Description API Include     */
#include <qliept.h>

/*********************************************************************/
/* Error Code Structure                                              */
/*                                                                   */
```

```
/* This shows how the user can define the variable length portion of   */
/* error code for the exception data.                                  */
/*                                                                     */
/*********************************************************************/
typedef struct {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
        } error_code_t;

main(int argc, char *argv[])
{
  error_code_t  error_code;
  char          qual_job_desc[20];
  char          *qual_job_ptr = qual_job_desc;
  char          rec_var[390];
  char          hold_value[10];
  char          message_id[7];
  char          command_string[53];
  char          message_string[67];

  memset(hold_value, ' ', 10);

  /*******************************************************************/
  /* Make sure we received the correct number of parameters. The argc  */
  /* parameter will contain the number of parameters that was passed   */
  /* to this program. This number also includes the program itself,    */
  /* so we need to evaluate argc-1.                                    */
  /*******************************************************************/

  if (((argc - 1) < 2) || ((argc - 1 > 2)))
  /*******************************************************************/
  /* We did not receive all of the required parameters so exit the     */
  /* program.                                                          */
  /*******************************************************************/
  {
    exit(1);
  }

  /*******************************************************************/
  /* Move the two parameter passed in into qual_job_desc.              */
  /*******************************************************************/
  memcpy(qual_job_ptr, argv[1], 10);
  qual_job_ptr += 10;
  memcpy(qual_job_ptr, argv[2], 10);

  /*******************************************************************/
  /* Set the error code parameter to 16.                               */
  /*******************************************************************/
  error_code.ec_fields.Bytes_Provided = 16;          (3)

  /*******************************************************************/
  /* Call the QWDRJOBD API.                                            */
  /*******************************************************************/
  QWDRJOBD(rec_var,               /* Receiver Variable                 */
           390,                   /* Receiver Length                   */
           "JOBD0100",            /* Format Name                       */
           qual_job_desc,         /* Qualified Job Description         */
           &error_code);          /* Error Code                        */

  /*******************************************************************/
  /* If an error was returned, send an error message.                  */
  /*******************************************************************/
  if(error_code.ec_fields.Bytes_Available > 0)        (2)
  {
    memcpy(message_id, error_code.ec_fields.Exception_Id, 7);
    sprintf(message_string,
            "SNDMSG MSG('Program failed with message ID %.7s') TOUSR(QPGMR)",
```

```
            message_id);
    system(message_string);
  }
  /*********************************************************************/
  /* Let's tell everyone what the hold value was for this job.         */
  /*********************************************************************/
  else
  {
    memcpy(hold_value, ((Qwd_JOBD0100_t *)rec_var)->Hold_Job_Queue, 10);
    sprintf(command_string,
            "SNDMSG MSG('HOLD value is %.10s') TOUSR(QPGMR)",
            hold_value);
    system(command_string);
  }

} /* main */
```

Related reference:

"Example in OPM RPG: Retrieving the HOLD parameter (error code structure)" on page 133
This OPM RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

Example in ILE RPG: Retrieving the HOLD parameter (error code structure)

This ILE RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
     D*****************************************************************
     D*****************************************************************
     D*
     D*  Program Name: JOBDAPI
     D*
     D*  Programming Language: ILE RPG
     D*
     D*  Description:  This program retrieves the HOLD value from
     D*                a job description.  It expects errors to be
     D*                returned via the error code parameter.
     D*
     D*  Header Files Included: QUSEC - Error Code Parameter
     D*                         QWDRJOBD - Retrieve Job Description API
     D*
     D*****************************************************************
     D*****************************************************************
     D*
     D* Error Code parameter include
     D*
     D/COPY QSYSINC/QRPGLESRC,QUSEC       (1)
     D*
     D* Retrieve Job Description API Include
     D*
     D/COPY QSYSINC/QRPGLESRC,QWDRJOBD
     D*
     D* Command string data structure
     D*
     DCMD_STRING       DS
     D                                26    INZ('SNDMSG MSG(''HOLD value is ')
     D HOLD                           10
     D                                15    INZ(''') TOUSR(QPGMR)')
     DCMD_STR2         DS
     D                                43    INZ('SNDMSG MSG(''Program failed -
     D                                      with message ID ')
     D MSG_ID                          7
```

```
D                                15    INZ(''') TOUSR(QPGMR)')
D*
D* Miscellaneous data structure
D*
DRCVLEN           S              9B 0 INZ(%SIZE(QWDD0100))
DFORMAT           S              8    INZ('JOBD0100')
DLENSTR           S             15  5 INZ(%SIZE(CMD_STRING))
DLENSTR2          S             15  5 INZ(%SIZE(CMD_STR2))
C*
C* Beginning of mainline
C*
C* Two parameters are being passed into this program
C*
C     *ENTRY        PLIST
C                   PARM                    JOBD            10
C                   PARM                    JOBD_LIB        10
C*
C* Move the two parameters passed into LFNAM
C*
C     JOBD          CAT       JOBD_LIB      LFNAM           20
C*
C* Error Code Bytes Provided is set to 16
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)     (3)
C*
C* Call the API.
C*
C                   CALL      QWDRJOBD
C                   PARM                    QWDD0100
C                   PARM                    RCVLEN
C                   PARM                    FORMAT
C                   PARM                    LFNAM
C                   PARM                    QUSEC
C*
C* Test for an error on the API call
C*
C                   IF        QUSBAVL > 0           (2)
C*
C* If there was an error, exit to ERROR subroutine
C*
C                   EXSR      ERROR
C*
C* Else, process the HOLD value
C*
C                   ELSE
C                   MOVEL     QWDHJQ        HOLD
C*
C* Let's tell everyone what the hold value was for this job
C*
C                   CALL      'QCMDEXC'
C                   PARM                    CMD_STRING
C                   PARM                    LENSTR
C                   END
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C*
C* End of MAINLINE
C*
C* Subroutine to handle errors received on the CALL
C*
C     ERROR         BEGSR
C*
C* Process errors returned from the API
C*
C                   MOVEL     QUSEI         MSG_ID
```

```
C                   CALL      'QCMDEXC'
C                   PARM                    CMD_STR2
C                   PARM                    LENSTR2
C                   ENDSR
```

Related reference:

"Example in OPM RPG: Retrieving the HOLD parameter (error code structure)" on page 133
This OPM RPG program retrieves the value of the HOLD parameter of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command). Any errors are returned in an error code structure.

Example in OPM RPG: Printing the HOLD value

This OPM RPG program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

This is the same type of program as the programs in "Example in OPM RPG: Retrieving the HOLD parameter (exception message)" on page 121 and "Example in OPM RPG: Retrieving the HOLD parameter (error code structure)" on page 133. The program, named JOBDAPI, prints the HOLD value if it is found, as shown at 1. If an error occurs, the program prints a line that contains the error message ID to a spooled file called QPRINT, as shown at 2.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*****************************************************************
F*****************************************************************
F*
F*Program Name: JOBDAPI
F*
F*Language:  OPM RPG
F*
F*Descriptive Name:  Get Job Description
F*
F*Description:  The following program prints out the name of
F*              the job description or prints an error if the
F*              API could not find the job description name
F*              specified.
F*
F*
F*Header Files Included: QUSEC - Error Code Parameter
F*                       QWDRJOBD - Retrieve Job Description API
F*
F*****************************************************************
F*****************************************************************
F* JOBDAPIR - Print value of HOLD parameter using API
F*    Uses error-code data structure
F*
FQPRINT  O   F     132     OF     PRINTER
I*
I* Error Code Parameter Include for the APIs
I*
I/COPY QSYSINC/QRPGSRC,QUSEC
I*
I* Retrieve Job Description API Include
I*
I/COPY QSYSINC/QRPGSRC,QWDRJOBD
I*
I*
I* Dummy data structure used to declare binary field          (3)
I*
I            DS
I I            390                   B   1   40RCVLEN
I I            'JOBD0100'                5   12FORMAT
C*
C* Beginning of Mainline
```

```
C*
C* Two parameters are being passed into this program.
C*
C           *ENTRY    PLIST                           Parm list
C                     PARM           JOBD   10        Job descrp
C                     PARM           JOBDL  10        Jobd library
C*
C* Move the two parameters passed into LFNAM.
C*
C           JOBD      CAT  JOBDL     LFNAM  20        Qlfd name
C*
C* Error code parameter is set to 16.
C*
C                     Z-ADD16        QUSBNB           Bytes provid
C*
C* Instead of specifying 'QWCRJOBD', I could have used the
C* constant QWDBGB that was defined in the QWDRJOBD include.
C* Call the API
C*
C                     CALL 'QWDRJOBD'                 Parm list
C                     PARM           QWDBH            Receiver Var.
C                     PARM           RCVLEN           Length RCVVAR
C                     PARM           FORMAT           Format Name
C                     PARM           LFNAM            Qual. Job Desc
C                     PARM           QUSBN            Error Code
C* If no bytes available, API was successful; print HOLD value
C           QUSBNC    IFEQ 0
C                     EXCPTGOOD
C                     ENDIF
C* If some bytes available, API failed; print error message ID
C           QUSBNC    IFGT 0
C                     EXCPTBAD
C                     ENDIF
C* End of program
C                     SETON                     LR
C                     RETRN
C*
C* End of MAINLINE
C*****************************************************************
O*
OQPRINT  E  106           GOOD
O                                    'HOLD value - '
O                         QWDBHN
OQPRINT  E  106           BAD
O                                    'Failed. Error ID - '
O                         QUSBND
```

The following data structures are used:

Error-code data structure

This defines the two binary fields used and the message ID that is returned for error conditions.

Retrieve job description data structure

This defines format JOBD0100, a 390-byte data structure with the hold field in positions 77-86.

Dummy data structure

This contains a field used for the length of the receiver variable. The field is defined as binary and is in the first 4 bytes. The dummy data structure, as shown at 3, also contains the format field.

This data structure is used because RPG only allows binary variables to be defined in the context of a data structure.

The program retrieves the parameter list that is passed and initializes the fields to be passed to the API. The API is called and places information into the receiver-variable data structure if information is found. The API places the information in the error-code data structure if an error occurred and if enough space was provided to receive the information.

The program prints one of two different lines depending on whether any errors were found:

```
HOLD value - *NO  (1)

Failed. Error ID - CPF9801  (2)
```

Related reference:

"Example in ILE COBOL: Printing the HOLD value"
This ILE COBOL program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

"Example in ILE C: Printing the HOLD value" on page 149
This ILE C program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

"Example in ILE RPG: Printing the HOLD value" on page 151
This ILE RPG program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

Example in ILE COBOL: Printing the HOLD value

This ILE COBOL program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

The following program also works for OPM COBOL.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*Program Name:          JOBDAPI
*
*Programming Language:  ILE COBOL
*
*Description:           This example shows how to print messages
*                       to spool files.
*
*Header Files Included: QUSEC - Error Code Parameter
*                       QWDRJOBD - Retrieve Job Description API
*
*****************************************************************
*****************************************************************
*
 PROGRAM-ID. JOBDAPI.
*
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
*
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
*
 DATA DIVISION.
 FILE SECTION.
*
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
```

```
*
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Retrieve Job Description API Include
*
 COPY QWDRJOBD OF QSYSINC-QLBLSRC.
*
* Command String Data Structure
*
 01  HOLD-VALUE.
     05  TEXT1 PIC X(13) VALUE 'HOLD value - '.
     05  HOLD  PIC X(10).
*
* Error Message Text
*
 01  MESSAGE-TEXT.
     05  MSG1   PIC X(19) VALUE 'Failed. Error ID - '.
     05  MSGID  PIC X(7).
*
 01  RECEIVER-LENGTH PIC S9(9) COMP-4.
 01  FORMAT-NAME PIC X(8) VALUE 'JOBD0100'.
 01  QCMDEXC PIC X(10) VALUE 'QCMDEXC'.
*
* Job Description and Library Name Structure
*
 01  JOBD-AND-LIB-NAME.
     05  JOB-DESC PIC X(10).
     05  JOB-DESC-LIB PIC X(10).
*
 LINKAGE SECTION.
*
* Two Parameters are being passed into this program.
*
 01  JOBD PIC X(10).
 01  JOBDL PIC X(10).
*
 PROCEDURE DIVISION USING JOBD, JOBDL.
 MAIN-LINE.
*
* Beginning of Mainline
*
* Move the two parameters passed into JOB-DESC and JOB-DESC-LIB.
*
     MOVE JOBD TO JOB-DESC.
     MOVE JOBDL TO JOB-DESC-LIB.
*
* Error Code Parameter is set to 16.
*
     MOVE 16 TO BYTES-PROVIDED.
*
* Receiver Length Set to 390.
*
     MOVE 390 TO RECEIVER-LENGTH.
*
* Call the QWDRJOBD API.
*
     CALL QWDRJOBD USING QWD-JOBD0100, RECEIVER-LENGTH,
```

```
                FORMAT-NAME, JOBD-AND-LIB-NAME, QUS-EC.
      *
      * If no bytes available, API was successful; print HOLD value
      *
        IF BYTES-AVAILABLE OF QUS-EC = 0 PERFORM GOOD.
      *
      * If some bytes available, API failed; print Error message ID
      *
        IF BYTES-AVAILABLE OF QUS-EC > 0 PERFORM BAD.

      *
           STOP RUN.
      *
      * End of Mainline
      *
      *
      * Subroutine to perform if no errors were encountered.
      *
       GOOD.

           OPEN OUTPUT LISTING.
           MOVE HOLD-JOB-QUEUE TO HOLD.
           WRITE LIST-LINE FROM HOLD-VALUE.

      *
      * Subroutine to perform if an error was returned in error code.
      *
       BAD.

           OPEN OUTPUT LISTING.
           MOVE EXCEPTION-ID TO MSGID.
           WRITE LIST-LINE FROM MESSAGE-TEXT.
           STOP RUN.
```

Related reference:

"Example in OPM RPG: Printing the HOLD value" on page 145
This OPM RPG program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

Example in ILE C: Printing the HOLD value

This ILE C program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*Program Name:          JOBDAPI                                     */
/*                                                                   */
/*Programming Language:  ILE C                                       */
/*                                                                   */
/*Description:           This example shows how to print messages    */
/*                       to spool files.                             */
/*                                                                   */
/*Header Files Included: STDIO - Standard Input/Output               */
/*                       STRING - String Functions                   */
/*                       QUSEC - Error Code Parameter                */
/*                       QWDRJOBD - Retrieve Job Description API     */
/*                       QLIEPT - Entry Point Table                  */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <stdio.h>
```

```
#include <string.h>
#include <qusec.h>        /* Error Code Parameter Include for the APIs */
#include <qwdrjobd.h>     /* Retrieve Job Description API Include      */
#include <qliept.h>       /* Entry Point Table Include                 */

/*****************************************************************/
/* Error Code Structure                                          */
/*                                                               */
/* This shows how the user can define the variable length portion of   */
/* error code for the exception data.                            */
/*                                                               */
/*****************************************************************/
typedef struct {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
        } error_code_t;

main(int argc, char *argv[])
{
  error_code_t  error_code;
  char          qual_job_desc[20];
  char          *qual_job_ptr = qual_job_desc;
  char          rec_var[390];
  char          hold_value[10];
  char          message_id[7];
  char          command_string[25];
  char          message_string[29];
  FILE          *stream;

  memset(hold_value, ' ', 10);

  /*****************************************************************/
  /* Make sure we received the correct number of parameters. The argc  */
  /* parameter will contain the number of parameters that was passed   */
  /* to this program. This number also includes the program itself,    */
  /* so we need to evaluate argc-1.                                */
  /*****************************************************************/

  if (((argc - 1) < 2) || ((argc - 1 > 2)))
  /*****************************************************************/
  /* We did not receive all of the required parameters so exit the     */
  /* program.                                                      */
  /*****************************************************************/
  {
    exit(1);
  }

  /*****************************************************************/
  /* Move the two parameter passed into qual_job_desc.             */
  /*****************************************************************/
  memcpy(qual_job_ptr, argv[1], 10);
  qual_job_ptr += 10;
  memcpy(qual_job_ptr, argv[2], 10);

  /*****************************************************************/
  /* Set the error code parameter to 16.                           */
  /*****************************************************************/
  error_code.ec_fields.Bytes_Provided = 16;

  /*****************************************************************/
  /* Open QPRINT file so that data can be written to it.  If the file  */
  /* cannot be opened, print a message and exit.                   */
  /*****************************************************************/
  if((stream = fopen("QPRINT", "wb")) == NULL)
  {
    printf("File could not be opened\n");
    exit(1);
```

```
  }

  /*********************************************************************/
  /* Call the QWDRJOBD API.                                            */
  /*********************************************************************/
  QWDRJOBD(rec_var,               /* Receiver Variable                 */
           390,                   /* Receiver Length                   */
           "JOBD0100",            /* Format Name                       */
           qual_job_desc,         /* Qualified Job Description         */
           &error_code);          /* Error Code                        */

  /*********************************************************************/
  /* If an error was returned, print the error message to the QPRINT   */
  /* spool file.                                                       */
  /*********************************************************************/
  if(error_code.ec_fields.Bytes_Available > 0)
  {
    memcpy(message_id, error_code.ec_fields.Exception_Id, 7);
    sprintf(message_string,
            "Failed.  Error ID - %.7s",
            message_id);
    fprintf(stream, message_string);
  }
  /*********************************************************************/
  /* Let's tell everyone what the hold value was for this job.         */
  /* The result will be printed in the QPRINT spool file.              */
  /*********************************************************************/
  else
  {
    memcpy(hold_value, ((Qwd_JOBD0100_t *)rec_var)->Hold_Job_Queue, 10);
    sprintf(command_string,
            "HOLD value - %.10s",
            hold_value);
    fprintf(stream, command_string);
  }

  fclose(stream);

} /* main */
```

Related reference:

"Example in OPM RPG: Printing the HOLD value" on page 145
This OPM RPG program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

Example in ILE RPG: Printing the HOLD value

This ILE RPG program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
     F*****************************************************************
     F*****************************************************************
     F*
     F*  Program Name: JOBDAPI
     F*
     F*  Programming Language: ILE RPG
     F*
     F*  Description:  This program retrieves the HOLD value from
     F*                a job description and then prints the value.
     F*                It expects errors to be returned via the
     F*                error code parameter.
     F*
     F*  Header Files Included: QUSEC - Error Code Parameter
     F*                         QWDRJOBD - Retrieve Job Description API
```

```
F*
F*************************************************************
F*************************************************************
F*
FQPRINT    O    F  132        PRINTER OFLIND(*INOF)
D*
D* Error Code parameter include
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D* Retrieve Job Description API Include
D*
D/COPY QSYSINC/QRPGLESRC,QWDRJOBD
D*
D* Miscellaneous data structure
D*
DRCVLEN           S              9B 0 INZ(%SIZE(QWDD0100))
DFORMAT           S              8    INZ('JOBD0100')
C*
C* Beginning of mainline
C*
C* Two parameters are being passed into this program
C*
C     *ENTRY        PLIST
C                   PARM                    JOBD            10
C                   PARM                    JOBD_LIB        10
C*
C* Move the two parameters passed into LFNAM
C*
C     JOBD          CAT       JOBD_LIB      LFNAM           20
C*
C* Error Code Bytes Provided is set to 16
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Call the API.
C*
C                   CALL      QWDRJOBD
C                   PARM                    QWDD0100
C                   PARM                    RCVLEN
C                   PARM                    FORMAT
C                   PARM                    LFNAM
C                   PARM                    QUSEC
C*
C* If no bytes available, API was successful; print HOLD value
C*
C                   IF        QUSBAVL = 0
C                   EXCEPT    GOOD
C                   ELSE
C*
C* If some bytes available, API failed; print Error message ID
C*
C                   IF        QUSBAVL > 0
C                   EXCEPT    BAD
C                   END
C                   END
C*
C* End of program
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C*
C* End of MAINLINE
C*************************************************************
O*
OQPRINT    E            GOOD           1  6
O                                              'HOLD value - '
```

```
O                       QWDHJQ
OQPRINT    E            BAD            1  6
O                                                 'Failed. Error ID - '
O                       QUSEI
```

Related reference:

"Example in OPM RPG: Printing the HOLD value" on page 145
This OPM RPG program prints the HOLD value of a job description (specified on the Create Job Description (CRTJOBD) or Change Job Description (CHGJOBD) command).

Example in OPM RPG: Accessing a field value (initial library list)

This OPM RPG program accesses a variable-length array. The variable-length array is the initial library list for a job description.

The discussion of the initial library list field in the job description format, JOBD0100 Format, indicates that the initial library list field is 11 bytes per entry, where each entry is a library name followed by a blank. Depending on how many libraries are named for the initial library list, the actual amount of space used varies (by multiples of 11).

The format does not have an entry in the *Offset* columns for initial library list. It might begin in offset 390, but do not rely on this offset value. For example, if a new field is added to the job description format, it will probably be placed at offset 390, and the initial library list information will be shifted.

To access the initial library list field, use the following fields in the format:

- Offset to the initial library list field, as shown at 1 in the program.
- Number of libraries in the initial library list field, as shown at 2.

If you use these field values in the format instead of statically encoding an offset and a number of libraries, your program can work on any future release of a business computing system, even if more job description attributes are defined in the format. This is an important approach to ensuring compatibility with future releases. Use this approach whenever you code for a list of entries.

The following RPG code sends a message for each library found in the initial library list field. Exceptions are handled by the RPG program. Although a library name cannot exceed 10 bytes, each entry is 11 bytes long.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
I*********************************************************************
I*********************************************************************
I*
I*Program Name: JOBDAPI
I*
I*Language:  OPM RPG
I*
I*Descriptive Name:  Get Job Description
I*
I*Description: This sample program shows the correct
I*             way of using the offset in a user space in RPG.
I*
I*Header Files Included: QUSEC - Error Code Parameter
I*                               (Copied into Program)
I*                       QWDRJOBD - Retrieve Job Description API
I*                                  (Copied into Program)
I*
I*********************************************************************
I*********************************************************************
I*
I* Error Code Parameter Include for the APIs
I*
```

```
I* The following QUSEC include is copied into this program
I* so that the variable-length field can be defined as
I* fixed length.
I*
I*
I*** START HEADER FILE SPECIFICATIONS ****************************
I*
I*Header File Name: H/QUSEC
I*
I*Descriptive Name: Error Code Parameter.
I*
I*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
I*All rights reserved.
I*US Government Users Restricted Rights -
I*Use, duplication or disclosure restricted
I*by GSA ADP Schedule Contract with IBM Corp.
I*
I*Licensed Materials-Property of IBM
I*
I*
I*Description: Include header file for the error code parameter.
I*
I*Header Files Included: None.
I*
I*Macros List: None.
I*
I*Structure List: Qus_EC_t
I*
I*Function Prototype List: None.
I*
I*Change Activity:
I*
I*CFD List:
I*
I*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
I*---- ------------ ----- ------ --------- ----------------------
I*$A0= D2862000     3D10  931201 DPOHLSON: New Include
I*
I*End CFD List.
I*
I*Additional notes about the Change Activity
I*End Change Activity.
I*** END HEADER FILE SPECIFICATIONS ******************************
I****************************************************************
I*Record structure for Error Code Parameter
I****                                                          ***
I*NOTE: The following type definition defines only the fixed
I*   portion of the format.  Varying-length field exception
I*   data is not defined here.
I****************************************************************
IQUSBN       DS
I*                                            Qus EC
I                                    B   1   40QUSBNB
I*                                            Bytes Provided
I                                    B   5   80QUSBNC
I*                                            Bytes Available
I                                        9  15 QUSBND
I*                                            Exception Id
I                                       16  16 QUSBNF
I*                                            Reserved
I*                                      Varying length, had to define len
I                                       17 100 QUSBNG
I*
I* Retrieve Job Description API Include
I*
I* The following QWDRJOBD include is copied into this program
I* so that the variable-length field can be defined as fixed
```

```
I* length.
I*
I*
I*** START HEADER FILE SPECIFICATIONS ****************************
I*
I*Header File Name: H/QWDRJOBD
I*
I*Descriptive Name: Retrieve Job Description Information API
I*
I*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
I*All rights reserved.
I*US Government Users Restricted Rights -
I*Use, duplication or disclosure restricted
I*by GSA ADP Schedule Contract with IBM Corp.
I*
I*Licensed Materials-Property of IBM
I*
I*
I*Description: The Retrieve Job Description Information API
I*          retrieves information from a job description
I*          object and places it into a single variable in the
I*          calling program.
I*
I*Header Files Included: None.
I*
I*Macros List: None.
I*
I*Structure List:  Qwd_JOBD0100_t
I*
I*Function Prototype List: QWDRJOBD
I*
I*Change Activity:
I*
I*CFD List:
I*
I*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
I*---- ------------ ----- ------ --------- ----------------------
I*$A0= D2862000     3D10  940424 ROCH:     New Include
I*
I*End CFD List.
I*
I*Additional notes about the Change Activity
I*End Change Activity.
I*** END HEADER FILE SPECIFICATIONS ******************************
I*****************************************************************
I*Prototype for QWDRJOBD API
I*****************************************************************
I              'QWDRJOBD'            C         QWDBGB
I*****************************************************************
I*Type Definition for the JOBD0100 format.
I****                                                          ***
I*NOTE: The following type definition defines only the fixed
I*   portion of the format.  Any varying-length fields have
I*   to be defined by the user.
I*****************************************************************
IQWDBH       DS                                5000
I*                                                Qwd JOBD0100
I                                      B   1   40QWDBHB
I*                                                Bytes Returned
I                                      B   5   80QWDBHC
I*                                                Bytes Available
I                                          9  18 QWDBHD
I*                                                Job Description Name
I                                         19  28 QWDBHF
I*                                                Job Description Lib Name
I                                         29  38 QWDBHG
I*                                                User Name
```

```
I                                    39  46 QWDBHH
I*                                      Job Date
I                                    47  54 QWDBHJ
I*                                      Job Switches
I                                    55  64 QWDBHK
I*                                      Job Queue Name
I                                    65  74 QWDBHL
I*                                      Job Queue Lib Name
I                                    75  76 QWDBHM
I*                                      Job Queue Priority
I                                    77  86 QWDBHN
I*                                      Hold Job Queue
I                                    87  96 QWDBHP
I*                                      Output Queue Name
I                                    97 106 QWDBHQ
I*                                      Output Queue Lib Name
I                                  107 108 QWDBHR
I*                                      Output Queue Priority
I                                  109 118 QWDBHS
I*                                      Printer Device Name
I                                  119 148 QWDBHT
I*                                      Print Text
I                                B 149 1520QWDBHV
I*                                      Syntax Check Severity
I                                B 153 1560QWDBHW
I*                                      End Severity
I                                B 157 1600QWDBHX
I*                                      Message Log Severity
I                                  161 161 QWDBHY
I*                                      Message Log Level
I                                  162 171 QWDBHZ
I*                                      Message Log Text
I                                  172 181 QWDBH0
I*                                      Log CL Programs
I                                  182 191 QWDBH1
I*                                      Inquiry Message Reply
I                                  192 204 QWDBH2
I*                                      Device Recovery Action
I                                  205 214 QWDBH3
I*                                      Time Slice End Pool
I                                  215 229 QWDBH4
I*                                      Accounting Code
I                                  230 309 QWDBH5
I*                                      Routing Data
I                                  310 359 QWDBH6
I*                                      Text Description
I                                  360 360 QWDBH7
I*                                      Reserved
I                                B 361 3640QWDBH8           (1)
I*                                      Offset Initial Lib List
I                                B 365 3680QWDBH9           (2)
I*                                      Number Libs In Lib list
I                                B 369 3720QWDBJB
I*                                      Offset Request Data
I                                B 373 3760QWDBJC
I*                                      Length Request Data
I                                B 377 3800QWDBJH
I*                                      Job Message Queue Max Size
I                                  381 390 QWDBJJ
I*                                      Job Message Queue Full Actio
I*                                 391 391 QWDBJD
I*
I*                             Varying length
I*                                 392 402 QWDBJF
I*
I*                                 Varying length
I*                                     403 403 QWDBJG
```

```
I*
I*
I* Command String Data Structure
I*
ICMDSTR      DS
I I            'SNDMSG MSG(''LIBRARY-    1  22 CMD1
I              ' - '
I                                       23  32 LIB
I I            ''') TOUSR(QPGMR)'       33  47 CMD2
I*
I* Miscellaneous Data Structure
I*
I            DS
I I            5000                  B   1   40RCVLEN
I I            0                     B   5   80X
I I            'JOBD0100'                9  16 FORMAT
C*
C* Beginning of Mainline
C*
C* Two parameters are being passed into this program.
C*
C           *ENTRY    PLIST
C                     PARM           JOBD   10
C                     PARM           JOBDL  10
C*
C* Move the two parameters passed into LFNAM.
C*
C           JOBD      CAT  JOBDL     LFNAM  20
C*
C* Error code Parameter is set to 100
C*
C                     Z-ADD100       QUSBNB
C*
C* Instead of specifying 'QWCRJOBD', I could have used the
C* constant QWDBGB that was defined in the QWDRJOBD include.
C*
C                     CALL 'QWDRJOBD'
C                     PARM           QWDBH            Receiver Var.
C                     PARM           RCVLEN           Length RCVVAR
C                     PARM           FORMAT           Format Name
C                     PARM           LFNAM            Qual. Job Desc
C                     PARM           QUSBN            Error Code
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C  N01                Z-ADD47        LENSTR 155
C*
C  N01      QWDBH8    ADD  1         X
C  N01      1         DO   QWDBH9
C           10        SUBSTQWDBH:X   LIB
C*
C* Let's tell everyone what the library value is.
C*
C                     CALL 'QCMDEXC'
C                     PARM           CMDSTR
C                     PARM           LENSTR
C                     ADD  11        X
C           X         IFGE RCVLEN
C                     LEAVE
C                     ENDIF
C                     ENDDO
C*
C                     SETON                     LR
C                     RETRN
C*
C* End of MAINLINE
C*
```

```
     C*
     C* Subroutine to handle errors returned in the error code
     C* parameter.
     C*
     C           ERRCOD    BEGSR
     C           QUSBNC    IFGT 0
     C                     SETON                     01    Error on API Call
     C*
     C* Process errors returned from the API.
     C*
     C                     END
     C                     ENDSR
```

Note: It is important to access the count and to compare for the exact number of libraries to be processed. If you do not check for the exact number of libraries, you may begin to access information in the format for the next set of information (in this example, it may be the request data value).

The output for this program example is as follows:

```
+------------------------------------------------------------------------------+
|                                                                              |
|                              Display Messages                                |
|                                                                              |
|                                                        System:   GENSYS90    |
| Queue . . . . . :   QPGMR                  Program . . . . :   *DSPMSG       |
|   Library . . . :     QUSRSYS                Library . . . :                 |
| Severity  . . . :   00                     Delivery  . . . :   *HOLD         |
| Type reply (if required), press Enter.                                       |
|   LIBRARY - SMITH                                                            |
|   From  . . . :   SMITH          07/23/94   12:29:38                         |
|   LIBRARY - QTEMP                                                            |
|   From  . . . :   SMITH          07/23/94   12:29:38                         |
|   LIBRARY - QGPL                                                             |
|   From  . . . :   SMITH          07/23/94   12:29:38                         |
|   LIBRARY - QBLDCPF                                                          |
|   From  . . . :   SMITH          07/23/94   12:29:38                         |
|   LIBRARY - UTIL                                                             |
|   From  . . . :   SMITH          07/23/94   12:29:38                         |
|   LIBRARY - OPENTEST                                                         |
+------------------------------------------------------------------------------+
```

The handling of the initial library list field is typical of what you will find in many APIs.

Related reference:

Retrieve Job Description Information (QWDRJOBD) API

"Example in ILE COBOL: Accessing a field value (initial library list)"
This ILE COBOL program accesses a variable-length array. The variable-length array is the initial library list for a job description.

"Example in ILE C: Accessing a field value (initial library list)" on page 162
This ILE C program accesses a variable-length array. The variable-length array is the initial library list for a job description.

"Example in ILE RPG: Accessing a field value (initial library list)" on page 165
This ILE RPG program accesses a variable-length array. The variable-length array is the initial library list for a job description.

Example in ILE COBOL: Accessing a field value (initial library list)

This ILE COBOL program accesses a variable-length array. The variable-length array is the initial library list for a job description.

The following program also works for OPM COBOL.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*Program Name:          JOBDAPI
*
*Programming Language:  COBOL
*
*Description:           This example shows how to access a
*                       field value returned from a retrieve
*                       API.
*
*Header Files Included: QUSEC - Error Code Parameter
*                       QWDRJOBD - Retrieve Job Description API
*
*****************************************************************
*****************************************************************
*
 PROGRAM-ID. JOBDAPI.
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Retrieve Job Description API Include
*
* The header file for the QWDRJOBD API was included in this
* program so that the varying length portion of the structure
* can be defined as a fixed portion.
*
*** START HEADER FILE SPECIFICATIONS ****************************
*
*Header File Name: H/QWDRJOBD
*
*Descriptive Name: Retrieve Job Description Information API
*
*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
*All rights reserved.
*US Government Users Restricted Rights -
*Use, duplication or disclosure restricted
*by GSA ADP Schedule Contract with IBM Corp.
*
*Licensed Materials-Property of IBM
*
*
*Description: The Retrieve Job Description Information API
*          retrieves information from a job description
*          object and places it into a single variable in the
*          calling program.
*
*Header Files Included: None.
*
*Macros List: None.
*
*Structure List:  Qwd_JOBD0100_t
```

```
*
*Function Prototype List: QWDRJOBD
*
*Change Activity:
*
*CFD List:
*
*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
*---- ------------ ----- ------ --------- ----------------------
*$A0= D2862000     3D10  940424 ROCH:     New Include
*
*End CFD List.
*
*Additional notes about the Change Activity
*End Change Activity.
*
*** END HEADER FILE SPECIFICATIONS *****************************
*****************************************************************
*Prototype for QWDRJOBD API
*****************************************************************
 77  QWDRJOBD                                PIC X(00010)
              VALUE "QWDRJOBD".
*****************************************************************
*Type Definition for the JOBD0100 format.
****                                                           ***
*NOTE: The following type definition defines only the fixed
*   portion of the format.  Any varying length field will
*   have to be defined by the user.
*****************************************************************
 01  RECEIVER-VARIABLE                       PIC  X(05000).
 01  QWD-JOBD0100 REDEFINES RECEIVER-VARIABLE.
     05  BYTES-RETURNED                      PIC S9(00009) BINARY.
     05  BYTES-AVAILABLE                     PIC S9(00009) BINARY.
     05  JOB-DESCRIPTION-NAME                PIC  X(00010).
     05  JOB-DESCRIPTION-LIB-NAME            PIC  X(00010).
     05  USER-NAME                           PIC  X(00010).
     05  JOB-DATE                            PIC  X(00008).
     05  JOB-SWITCHES                        PIC  X(00008).
     05  JOB-QUEUE-NAME                      PIC  X(00010).
     05  JOB-QUEUE-LIB-NAME                  PIC  X(00010).
     05  JOB-QUEUE-PRIORITY                  PIC  X(00002).
     05  HOLD-JOB-QUEUE                      PIC  X(00010).
     05  OUTPUT-QUEUE-NAME                   PIC  X(00010).
     05  OUTPUT-QUEUE-LIB-NAME               PIC  X(00010).
     05  OUTPUT-QUEUE-PRIORITY               PIC  X(00002).
     05  PRINTER-DEVICE-NAME                 PIC  X(00010).
     05  PRINT-TEXT                          PIC  X(00030).
     05  SYNTAX-CHECK-SEVERITY               PIC S9(00009) BINARY.
     05  END-SEVERITY                        PIC S9(00009) BINARY.
     05  MESSAGE-LOG-SEVERITY                PIC S9(00009) BINARY.
     05  MESSAGE-LOG-LEVEL                   PIC  X(00001).
     05  MESSAGE-LOG-TEXT                    PIC  X(00010).
     05  LOG-CL-PROGRAMS                     PIC  X(00010).
     05  INQUIRY-MESSAGE-REPLY               PIC  X(00010).
     05  DEVICE-RECOVERY-ACTION              PIC  X(00013).
     05  TIME-SLICE-END-POOL                 PIC  X(00010).
     05  ACCOUNTING-CODE                     PIC  X(00015).
     05  ROUTING-DATA                        PIC  X(00080).
     05  TEXT-DESCRIPTION                    PIC  X(00050).
     05  RESERVED                            PIC  X(00001).
     05  OFFSET-INITIAL-LIB-LIST             PIC S9(00009) BINARY.     (1)
     05  NUMBER-LIBS-IN-LIB-LIST             PIC S9(00009) BINARY.     (2)
     05  OFFSET-REQUEST-DATA                 PIC S9(00009) BINARY.
     05  LENGTH-REQUEST-DATA                 PIC S9(00009) BINARY.
     05  JOB-MESSAGE-QUEUE-MAX-SIZE          PIC S9(00009) BINARY.
     05  JOB-MESSAGE-QUEUE-FULL-ACTION       PIC  X(00010).
*    05  RESERVED2                           PIC  X(00001).
```

```
*
*                             Varying length
*    05  INITIAL-LIB-LIST                   PIC  X(00011).
*
*
*                             Varying length
*    05  REQUEST-DATA                       PIC  X(00001).
*
*                             Varying length
*
* Command String Data Structure
*
 01  COMMAND-STRING.
     05  TEXT1 PIC X(22) VALUE 'SNDMSG MSG(''LIBRARY- '.
     05  LIB   PIC X(10).
     05  TEXT2 PIC X(15) VALUE ''') TOUSR(QPGMR)'.
*
 01  COMMAND-LENGTH PIC S9(10)V99999 COMP-3.
 01  RECEIVER-LENGTH PIC S9(9) COMP-4.
 01  FORMAT-NAME PIC X(8) VALUE 'JOBD0100'.
 01  QCMDEXC PIC X(10) VALUE 'QCMDEXC'.
 01  X PIC S9(9) BINARY.
*
* Job Description and Library Name Structure
*
 01  JOBD-AND-LIB-NAME.
     05  JOB-DESC PIC X(10).
     05  JOB-DESC-LIB PIC X(10).
*
 LINKAGE SECTION.
*
* Two Parameters are being passed into this program.
*
 01  JOBD PIC X(10).
 01  JOBDL PIC X(10).
*
 PROCEDURE DIVISION USING JOBD, JOBDL.
 MAIN-LINE.
*
* Beginning of Mainline
*
* Move the two parameters passed into JOB-DESC and JOB-DESC-LIB.
*
     MOVE JOBD TO JOB-DESC.
     MOVE JOBDL TO JOB-DESC-LIB.
*
* Error Code Parameter is set to 100.
*
     MOVE 100 TO BYTES-PROVIDED.
*
* Receiver Length Set to 5000.
*
     MOVE 5000 TO RECEIVER-LENGTH.
*
* Call the QWDRJOBD API.
*
     CALL QWDRJOBD USING RECEIVER-VARIABLE, RECEIVER-LENGTH,
             FORMAT-NAME, JOBD-AND-LIB-NAME, QUS-EC.
*
* See if any errors were returned in the error code parameter.
*
     PERFORM ERRCOD.
*
* Add one to the Initial library list offset because COBOL is a
* Base 1 language.
*
     MOVE OFFSET-INITIAL-LIB-LIST TO X.
```

```
           ADD 1 TO X.
           MOVE 47 TO COMMAND-LENGTH.
      *
      * Let's tell everyone what the library value was for this job.
      *
           PERFORM NUMBER-LIBS-IN-LIB-LIST TIMES
                   MOVE RECEIVER-VARIABLE(X:10) TO LIB,
                   CALL QCMDEXC USING COMMAND-STRING, COMMAND-LENGTH,
                   ADD 11 TO X,
                   PERFORM RECLEN,
                   END-PERFORM.

      *
           STOP RUN.
      *
      * End of Mainline
      *
      *
      * Subroutine to handle errors returned in the error code
      * parameter.
      *
       ERRCOD.
      *
           IF BYTES-AVAILABLE OF QUS-EC > 0
      *
      * Process errors returned from the API.
      *
             STOP RUN.
      *
      * Subroutine to check to see if there is enough room in the
      * receiver variable for the next library in the list.
      *
       RECLEN.
      *
             IF (X + 10) >= RECEIVER-LENGTH
               STOP RUN.
```

Related reference:

"Example in OPM RPG: Accessing a field value (initial library list)" on page 153
This OPM RPG program accesses a variable-length array. The variable-length array is the initial library list for a job description.

Example in ILE C: Accessing a field value (initial library list)

This ILE C program accesses a variable-length array. The variable-length array is the initial library list for a job description.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*Program Name:          JOBDAPI                                     */
/*                                                                   */
/*Programming Language:  ILE C                                       */
/*                                                                   */
/*Description:           This example shows how to access a field    */
/*                       value returned from a retrieve API.         */
/*                                                                   */
/*Header Files Included: STDIO - Standard Input/Output               */
/*                       STRING - String Functions                   */
/*                       QUSEC - Error Code Parameter                */
/*                       QWDRJOBD - Retrieve Job Description API     */
/*                       QLIEPT - Entry Point Table                  */
/*                                                                   */
/*********************************************************************/
```

```
/*********************************************************************/

#include <stdio.h>
#include <string.h>
#include <qusec.h>        /* Error Code Parameter Include for the APIs */
#include <qwdrjobd.h>     /* Retrieve Job Description API Include      */
#include <qliept.h>       /* Entry Point Table Include                 */

/*********************************************************************/
/* Error Code Structure                                              */
/*                                                                   */
/* This shows how the user can define the variable-length portion of */
/* error code for the exception data.                                */
/*                                                                   */
/*********************************************************************/
typedef struct {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
        } error_code_t;

/*********************************************************************/
/* JOBD0100 Structure                                                */
/*                                                                   */
/* This shows how the user can define the variable-length portion of */
/* the JOBD0100 format.                                              */
/*                                                                   */
/*********************************************************************/
typedef struct {
     Qwd_JOBD0100_t data;
     char           Lib_Data[5000];  (1)  (2)
        } JOBD0100;

main(int argc, char *argv[])
{
  error_code_t  error_code;
  char          library[10];
  char          qual_job_desc[20];
  char          *qual_job_ptr = qual_job_desc;
  char          rec_var[1000];
  char          *rec_ptr = rec_var;
  char          hold_value[10];
  char          message_id[7];
  char          command_string[49];
  int           i;
  int           num_libs;
  int           offset;
  int           rec_len = 5000;

  memset(hold_value, ' ', 10);

  /*******************************************************************/
  /* Make sure we received the correct number of parameters. The argc  */
  /* parameter will contain the number of parameters that was passed   */
  /* to this program. This number also includes the program itself,    */
  /* so we need to evaluate argc-1.                                    */
  /*******************************************************************/

  if (((argc - 1) < 2) || ((argc - 1 > 2)))
  /*******************************************************************/
  /* We did not receive all of the required parameters so exit the     */
  /* program.                                                          */
  /*******************************************************************/
  {
    exit(1);
  }

  /*******************************************************************/
```

```
  /* Move the two parameter passed into qual_job_desc.                */
  /*********************************************************************/
  memcpy(qual_job_ptr, argv[1], 10);
  qual_job_ptr += 10;
  memcpy(qual_job_ptr, argv[2], 10);

  /*********************************************************************/
  /* Set the error code parameter to 16.                               */
  /*********************************************************************/
  error_code.ec_fields.Bytes_Provided = 16;

  /*********************************************************************/
  /* Call the QWDRJOBD API.                                            */
  /*********************************************************************/
  QWDRJOBD(rec_var,               /* Receiver Variable                 */
           rec_len,               /* Receiver Length                   */
           "JOBD0100",            /* Format Name                       */
           qual_job_desc,         /* Qualified Job Description         */
           &error_code);          /* Error Code                        */

  /*********************************************************************/
  /* If an error was returned, send an error message.                  */
  /*********************************************************************/
  if(error_code.ec_fields.Bytes_Available > 0)
  {
    /* In this example, nothing was done for the error condition.      */
  }
  /*********************************************************************/
  /* Let's tell everyone what the library value was for this job.      */
  /*********************************************************************/
  else
  {
    num_libs = ((JOBD0100 *)rec_var)->data.Number_Libs_In_Lib_list;
    offset = ((JOBD0100 *)rec_var)->data.Offset_Initial_Lib_List;

    /*******************************************************************/
    /* Advance receiver variable pointer to the location where the     */
    /* library list begins.                                            */
    /*******************************************************************/
    rec_ptr += offset;

    for(i=0; i<num_libs; i++)
    {
      memcpy(library, rec_ptr, 10);
      sprintf(command_string,
              "SNDMSG MSG('LIBRARY %.10s') TOUSR(QPGMR)",
               library);
      system(command_string);

      rec_ptr += 11;
      if((offset + 10) >= rec_len)
        break;
      offset += 11;
    }

  }

} /* main */
```

Related reference:

"Example in OPM RPG: Accessing a field value (initial library list)" on page 153
This OPM RPG program accesses a variable-length array. The variable-length array is the initial library list for a job description.

Example in ILE RPG: Accessing a field value (initial library list)

This ILE RPG program accesses a variable-length array. The variable-length array is the initial library list for a job description.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
D*****************************************************************
D*****************************************************************
D*
D*  Program Name: JOBDAPI
D*
D*  Programming Language: ILE RPG
D*
D*  Description:  This program retrieves the library list from
D*                a job description.  It expects errors to be
D*                returned via the error code parameter.
D*
D*  Header Files Included: QUSEC - Error Code Parameter
D*
D*  Header Files Modified: QWDRJOBD - Retrieve Job Description API
D*
D*****************************************************************
D*****************************************************************
D*
D* Error Code parameter include
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D* The following QWDRJOBD include from QSYSINC is copied into
D* this program so that it can be declared as 1000 bytes in
D* size.  This size should accommodate the variable length Library
D* List array.
D*
D*** START HEADER FILE SPECIFICATIONS ****************************
D*
D*Header File Name: H/QWDRJOBD
D*
D*Descriptive Name: Retrieve Job Description Information API
D*
D*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
D*All rights reserved.
D*US Government Users Restricted Rights -
D*Use, duplication or disclosure restricted
D*by GSA ADP Schedule Contract with IBM Corp.
D*
D*Licensed Materials-Property of IBM
D*
D*
D*Description: The Retrieve Job Description Information API
D*          retrieves information from a job description
D*          object and places it into a single variable in the
D*          calling program.
D*
D*Header Files Included: None.
D*
D*Macros List: None.
D*
D*Structure List:  Qwd_JOBD0100_t
D*
D*Function Prototype List: QWDRJOBD
D*
D*Change Activity:
D*
D*CFD List:
D*
```

```
D*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
D*---- ------------ ----- ------ --------- ----------------------
D*$A0= D2862000     3D10  940424 ROCH:     New Include
D*
D*End CFD List.
D*
D*Additional notes about the Change Activity
D*End Change Activity.
D*** END HEADER FILE SPECIFICATIONS *****************************
D*****************************************************************
D*Prototype for QWDRJOBD API
D*****************************************************************
D QWDRJOBD        C                   'QWDRJOBD'
D*****************************************************************
D*Type Definition for the JOBD0100 format.
D****                                                          ***
D*NOTE: The following type definition defines only the fixed
D*   portion of the format.  Any varying length field will
D*   have to be defined by the user.
D*****************************************************************
DQWDD0100         DS          5000
D*                                          Qwd JOBD0100
D QWDBRTN                 1      4B 0
D*                                          Bytes Returned
D QWDBAVL                 5      8B 0
D*                                          Bytes Available
D QWDJDN                  9     18
D*                                          Job Description Name
D QWDJDLN                19     28
D*                                          Job Description Lib Name
D QWDUN                  29     38
D*                                          User Name
D QWDJD                  39     46
D*                                          Job Date
D QWDJS                  47     54
D*                                          Job Switches
D QWDJQN00               55     64
D*                                          Job Queue Name
D QWDJQLN00              65     74
D*                                          Job Queue Lib Name
D QWDJQP                 75     76
D*                                          Job Queue Priority
D QWDHJQ                 77     86
D*                                          Hold Job Queue
D QWDOQN                 87     96
D*                                          Output Queue Name
D QWDOQLN                97    106
D*                                          Output Queue Lib Name
D QWDOQP                107    108
D*                                          Output Queue Priority
D QWDPDN                109    118
D*                                          Printer Device Name
D QWDPT                 119    148
D*                                          Print Text
D QWDSCS                149    152B 0
D*                                          Syntax Check Severity
D QWDES                 153    156B 0
D*                                          End Severity
D QWDMLS                157    160B 0
D*                                          Message Log Severity
D QWDMLL                161    161
D*                                          Message Log Level
D QWDMLT                162    171
D*                                          Message Log Text
D QWDLCLP               172    181
D*                                          Log CL Programs
D QWDIMR                182    191
```

```
D*                                              Inquiry Message Reply
D QWDDRA                192    204
D*                                              Device Recovery Action
D QWDTSEP               205    214
D*                                              Time Slice End Pool
D QWDAC                 215    229
D*                                              Accounting Code
D QWDRD                 230    309
D*                                              Routing Data
D QWDTD                 310    359
D*                                              Text Description
D QWDERVED00            360    360
D*                                              Reserved
D QWDOILL               361    364B 0               (1)
D*                                              Offset Initial Lib List
D QWDNLILL              365    368B 0               (2)
D*                                              Number Libs In Lib list
D QWDORD                369    372B 0
D*                                              Offset Request Data
D QWDLRD                373    376B 0
D*                                              Length Request Data
D QWDJMQMS              377    380B 0
D*                                              Job Message Queue Max Size
D QWDJMQFA              381    390
D*                                              Job Msg Queue Full Action
D*QWDRSV2               391    391
D*
D*                              Varying length
D*QWDILL                392    402    DIM(00001)
D*
D*                                  Varying length
D*QWDRD00               403    403
D*
D*                              Varying length
D*
D* Command string data structure
D*
DCMD_STRING       DS
D                                22    INZ('SNDMSG MSG(''LIBRARY - ')
D LIBRARY                        10
D                                15    INZ(''') TOUSR(QPGMR)')
D*
D* Miscellaneous data structure
D*
DRCVLEN           S               9B 0 INZ(%SIZE(QWDD0100))
DFORMAT           S               8    INZ('JOBD0100')
DLENSTR           S              15  5 INZ(%SIZE(CMD_STRING))
C*
C* Beginning of mainline
C*
C* Two parameters are being passed into this program
C*
C     *ENTRY        PLIST
C                   PARM                    JOBD            10
C                   PARM                    JOBD_LIB        10
C*
C* Move the two parameters passed into LFNAM
C*
C     JOBD          CAT       JOBD_LIB      LFNAM           20
C*
C* Error Code Bytes Provided is set to 16
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Call the API.
C*
C                   CALL      QWDRJOBD
```

```
C                   PARM                    QWDD0100
C                   PARM                    RCVLEN
C                   PARM                    FORMAT
C                   PARM                    LFNAM
C                   PARM                    QUSEC
C*
C* Test for an error on the API call
C*
C                   IF        QUSBAVL > 0
C*
C* If there was an error, exit to ERROR subroutine
C*
C                   EXSR      ERROR
C                   ELSE
C*
C* Else, add 1 to the Initial library list offset because RPG
C* is a Base 1 language
C*
C     QWDOILL       ADD       1             X                 5 0
C                   DO        QWDNLILL
C                   EVAL      LIBRARY = %SUBST(QWDD0100:X:10)
C*
C* Let's tell everyone what the library value is
C*
C                   CALL      'QCMDEXC'
C                   PARM                    CMD_STRING
C                   PARM                    LENSTR
C                   ADD       11            X
C                   IF        (X + 10) > RCVLEN
C                   LEAVE
C                   ENDIF
C                   ENDDO
C                   ENDIF
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C*
C* End of MAINLINE
C*
C* Subroutine to handle errors returned in the error code parameter
C*
C     ERROR         BEGSR
C*
C* Process errors returned from the API.  As this sample program
C* used /COPY to include the error code structure, only the first
C* 16 bytes of the error code structure are available.  If the
C* application program needed to access the variable length
C* exception data for the error, the developer should physically
C* copy the QSYSINC include and modify the copied include to
C* define additional storage for the exception data.
C*
C                   ENDSR
```

Related reference:

"Example in OPM RPG: Accessing a field value (initial library list)" on page 153
This OPM RPG program accesses a variable-length array. The variable-length array is the initial library list for a job description.

Example in OPM RPG: Using keys with the List Spooled Files (QUSLSPL) API

This OPM RPG program processes a list of spooled file information that you have specified using keys.

This example introduces a program named LSTSPL. Program LSTSPL uses the List Spooled Files (QUSLSPL) API to determine the spooled file name, date created, and number of pages for all spooled files that are created by the current user of the LSTSPL program.

Unlike the earlier JOBDAPI program examples, where format JOBD0100 of the Retrieve Job Description (QWDRJOBD) API returned dozens of fields while we were only interested in the HOLD field, the QUSLSPL API provides a keyed interface that allows LSTSPL to request that only the relevant fields (spooled file name, date created, and number of pages) be returned.

In addition to providing a keyed interface, QUSLSPL also differs from QWDRJOBD in that the QUSLSPL API retrieves a list of all spooled files into a User Space (*USRSPC) while QWDRJOBD retrieves information about one specific job description into a program variable.

In the following program example, all the pieces have been put together with an OPM RPG program that accesses specific information related to spooled files. A report listing this information is created. The program example does not handle API-related errors. Any errors that are received are returned as exception messages, as shown at 1.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*************************************************************
F*
F* Program Name:          LSTSPL
F*
F* Program Language:      OPM RPG
F*
F* Descriptive Name:      List Spooled Files for Current User
F*
F* Description:           This example shows the steps necessary
F*                        to process keyed output from an API.
F*
F* Header Files Included: QUSEC    - Error Code Parameter
F*                        QUSGEN   - User Space Generic Header
F*                        QUSLSPL  - List Spooled Files
F*
F* APIs Used:             QUSLSPL  - List Spooled Files
F*                        QUSCRTUS - Create User Space
F*                        QUSRTVUS - Retrieve User Space
F*
F*************************************************************
FQSYSPRT O   F     132     OF     PRINTER
I*
I* Copy User Space Generic Header
I*
I/COPY QSYSINC/QRPGSRC,QUSGEN       (11)
I*
I* Copy API Error Code parameter
I*
I/COPY QSYSINC/QRPGSRC,QUSEC
I*
I* Copy List Spooled Files API include
I*
I/COPY QSYSINC/QRPGSRC,QUSLSPL
I*
I* Data structure to hold space name
I*
ISPCNAM      DS
I I            'SPCNAME   '              1  10 SPC
I I            'QTEMP     '             11  20 LIB
I*
I* Data structure to hold requested key values
I*
IKEYARA      DS     (7)
I I            201                   B   1   40KEY1
I I            216                   B   5   80KEY2
I I            211                   B   9  120KEY3    (8)
I*
I* Receiver variable for QUSRTVUS
```

```
I*
IRECVR       DS                          1000
I*
I* Other assorted variables
I*
I            DS
I                                    B   1   40SIZ
I                                    B   5   80START
I                                    B   9  120LENDTA
I                                    B  13  160KEY#
I                                    B  17  200PAGES#
I                                       17  20 PAGESA
I I            X'00'                    21  21 INTVAL
C*
C* Initialize Error Code structure to accept exceptions
C*
C                     Z-ADD0         QUSBNB   (1)
C*
C* Create the User Space to hold the QUSLSPL API results
C*
C                     CALL 'QUSCRTUS'    (2)
C                     PARM           SPCNAM
C                     PARM 'quslspl' EXTATR 10
C                     PARM 2000      SIZ
C                     PARM           INTVAL
C                     PARM '*ALL'    PUBAUT 10
C                     PARM           TXTDSC 50
C                     PARM '*YES'    REPLAC 10
C                     PARM           QUSBN
C*
C* Call QUSLSPL to get all spooled files for *CURRENT user
C*
C                     CALL 'QUSLSPL'     (3)
C                     PARM           SPCNAM
C                     PARM 'SPLF0200'FORMAT  8   (4)
C                     PARM '*CURRENT'USRNAM 10
C                     PARM '*ALL'    OUTQ   20
C                     PARM '*ALL'    FRMTYP 10
C                     PARM '*ALL'    USRDTA 10
C                     PARM           QUSBN
C                     PARM           JOBNAM 26
C                     PARM           KEYARA      (5)
C                     PARM 3         KEY#        (6)
C*
C* Retrieve information concerning the User Space and its contents
C*
C                     CALL 'QUSRTVUS'    (9)
C                     PARM           SPCNAM
C                     PARM 1         START            Start Rtv at 1
C                     PARM 192       LENDTA           for length =192
C                     PARM           QUSBP      (10)
C                     PARM           QUSBN
C*
C* Check User Space status for good information
C*
C           QUSBPD    IFEQ '0100'       (12)    Header Fmt
C           QUSBPJ    IFEQ 'C'          (14)    Complete
C           QUSBPJ    OREQ 'P'                        or Partial
C*
C* Check to see if any entries were put into User Space
C*
C           QUSBPS    IFGT 0            (16)
C*
C* Keep count of how many list entries we have processed
C*
C                     Z-ADD0         COUNT   90   (17)
C*
```

```
C* Adjust Offset value to Position value
C*
C           QUSBPQ    ADD  1         START       (18)
C*
C* Retrieve the lesser of allocated storage or available data
C*
C           QUSBPT    IFLT 1000         (19)
C                     Z-ADDQUSBPT    LENDTA
C                     ELSE
C                     Z-ADD1000      LENDTA
C                     ENDIF
C*
C* Process all entries returned
C*
C           COUNT     DOWLTQUSBPS        (20)
C*
C* Retrieve spooled file information
C*
C                     CALL 'QUSRTVUS'   (21)
C                     PARM           SPCNAM
C                     PARM           START
C                     PARM           LENDTA
C                     PARM           RECVR
C                     PARM           QUSBN
C*
C* Loop through returned fields
C*
C           4         SUBSTRECVR     QUSFV        (22)
C                     Z-ADD5         X       40
C                     DO   QUSFVB        (23)
C*
C* Get header information
C*
C           16        SUBSTRECVR:X   QUSKR        (24)
C*
C* Set Y to location of actual data associated with key
C*
C           X         ADD  16        Y       40
C*
C* Process the data based on key type
C*
C           QUSKRC    CASEQ201       FILNAM        (25)
C           QUSKRC    CASEQ211       PAGES
C           QUSKRC    CASEQ216       AGE
C                     CAS            ERROR
C                     END
C*
C* Adjust X to address next keyed record returned
C*
C                     ADD  QUSKRB    X
C                     ENDDO
C*
C* Output information on spooled file
C*
C                     EXCPTPRTLIN                (26)
C*
C* Adjust START to address next entry
C*
C                     ADD  1         COUNT        (27)
C                     ADD  QUSBPT    START
C                     ENDDO
C                     ENDIF
C                     ELSE               (15)
C                     EXCPTLSTERR
C                     ENDIF
C                     ELSE               (13)
C                     EXCPTHDRERR
```

```
C                     ENDIF
C                     MOVE '1'       *INLR        (28)
C                     RETRN
C*
C* Various subroutines
C*
C*************************************************************
C           FILNAM    BEGSR
C*
C* Extract spooled file name for report
C*
C                     MOVE *BLANKS   PRTFIL 10
C           QUSKRG    SUBSTRECVR:Y   PRTFIL
C                     ENDSR
C*************************************************************
C           PAGES     BEGSR
C*
C* Extract number of pages for report
C*
C           QUSKRG    SUBSTRECVR:Y   PAGESA
C                     ENDSR
C*************************************************************
C           AGE       BEGSR
C*
C* Extract age of spooled file for report
C*
C                     MOVE *BLANKS   OPNDAT  7
C           QUSKRG    SUBSTRECVR:Y   OPNDAT
C                     ENDSR
C*************************************************************
C           ERROR     BEGSR
C*
C* If unknown key value, then display the value and end
C*
C                     DSPLY          QUSKRC
C                     MOVE '1'       *INLR
C                     RETRN
C                     ENDSR
O*
OQSYSPRT E               PRTLIN
O                        PRTFIL    10
O                        PAGES#    25
O                        OPNDAT    40
OQSYSPRT E               LSTERR
O                                  22 'List data not valid   '
OQSYSPRT E               HDRERR
O                                  22 'Unknown Generic Header'
```

List APIs do not automatically create the user space (*USRSPC) to receive the list. You must first create one using the Create User Space (QUSCRTUS) API (2). Similar to CL create commands, the QUSCRTUS API has several parameters that identify the name of the object, the public authority, the object description text, and so forth.

After creating the user space, you can call the QUSLSPL API to return spooled file information into the user space (3). The QUSLSPL API supports two formats: SPLF0100, which returns a fixed set of information about each selected spooled file, and SPLF0200, which returns only user-selected fields. LSTSPL uses SPLF0200 (4) and passes to the QUSLSPL API a list of keys to identify the selected fields (5) and the number of keys (6). Because OPM RPG does not support an array (list) of binary values, LSTSPL defines the key array (KEYARA) as a data structure comprised of contiguous binary(4) fields (7). The fields are initialized to 201, 216, and 211, which correspond to the keys named spooled file name, date file was opened, and total pages, respectively (8). Note that while the user space was created with an initial size of 2000 bytes (2), most List APIs implicitly extend the user space (up to a maximum of 16MB) in order to return all available list entries. The reverse, truncation when the user space is too large, is not performed by list APIs.

Having generated the list, you can now process the user space data.

List APIs (like QUSLSPL) generally provide a generic list header at the beginning of the user space, which provides information such as the API that created the list, the number of entries (spooled files for this example) in the list, the size of each entry, and so on. To access the generic list header, use the Retrieve User Space (QUSRTVUS) API (9). Program LSTSPL retrieves the generic list header into the data structure QUSBP (10), which is defined in the QUSGEN QSYSINC /COPY (include) file (11). Note that languages, such as ILE RPG, COBOL, and C, which support pointers, can avoid this call to QUSRTVUS (and the resulting movement of data) by using the Retrieve Pointer to User Space (QUSPTRUS) API.

Program LSTSPL now checks that the format of the generic list header is the one expected (12), and if not, prints an error line (13). Having verified the header format, LSTSPL now checks the information status of the list (14) (and if it is not accurate, prints an error line (15)) and that at least one list entry is available (16).

Having determined that accurate list entries are available, program LSTSPL performs the following operations:

- Initialize the COUNT variable to keep track of how many entries have been processed (17).
- Add one to the base 0 offset (to the first entry in the list) as the QUSRTVUS API assumes base 1 positional values (18).
- Determine how much data is associated with each entry (19) (which is the lesser of either the amount of storage you allocated to receive a list entry or the size of a list entry).
- Fall into a DO loop to process all of the available list entries (20).

Within this loop, LSTSPL retrieves each list entry (21), extracts the number of fields returned (22), and enters an inner DO loop to process all of the available list entry fields (23).

Within this inner loop, the program extracts the field information (24) and processes the field data based on the key field (25).

When all fields for a given list entry have been processed, LSTSPL generates a print line (26) and proceeds to the next list entry (27).

When all the list entries have been processed, LSTSPL ends (28).

Related concepts:

"Receiver variables" on page 73
A *receiver variable* is a program variable that is used as an output field to contain information that is returned from a retrieve API.

"List APIs overview" on page 76
List APIs return a list of information. They use a common generic header to provide information such as the number of list entries and the size of a list entry. The content of the list is unique to each API.

Related reference:

"Example in ILE COBOL: Using keys with the List Spooled Files (QUSLSPL) API" on page 174
This ILE COBOL program processes a list of spooled file information that you have specified using keys.

"Example in ILE C: Using keys with the List Spooled Files (QUSLSPL) API" on page 177
This ILE C program processes a list of spooled file information that you have specified using keys.

"Example in ILE RPG: Using keys with the List Spooled Files (QUSLSPL) API" on page 181
This ILE RPG program processes a list of spooled file information that you have specified using keys.

"User spaces" on page 67
List APIs return information to user spaces. A *user space* is an object consisting of a collection of bytes that can be used for storing any user-defined information.

Example in ILE COBOL: Using keys with the List Spooled Files (QUSLSPL) API

This ILE COBOL program processes a list of spooled file information that you have specified using keys.

The following program also works for OPM COBOL.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*  Program:      List Spooled Files for Current User
*
*  Language:     ILE COBOL
*
*  Description:  This example shows the steps necessary to
*                process keyed output from an API.
*
*  APIs Used:    QUSLSPL  - List Spooled Files
*                QUSCRTUS - Create User Space
*                QUSPTRUS - Retrieve Pointer to User Space
*
*****************************************************************
*****************************************************************
*
 PROGRAM-ID. LSTSPL.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION.
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
*
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Listing text
*
 01  PRTLIN.
     05  PRTFIL        PIC  X(10).
     05  FILLER        PIC  X(05).
     05  PAGES         PIC S9(09).
     05  FILLER        PIC  X(05).
     05  OPNDAT        PIC  X(07).
 01  LSTERR.
     05  TEXT1         PIC  X(22) VALUE "List data not valid".
 01  HDRERR.
     05  TEXT2         PIC  X(22) VALUE "Unknown Generic Header".
*
 01  MISC.
     05  SPC-NAME      PIC  X(20) VALUE "SPCNAME   QTEMP     ".
     05  SPC-SIZE      PIC S9(09) VALUE 2000 BINARY.   (2)
```

```
     05  SPC-INIT      PIC  X(01) VALUE X"00".
     05  SPCPTR        POINTER.
     05  SPC-TYPE      PIC  X(10) VALUE "*USRSPC".
     05  EXT-ATTR      PIC  X(10) VALUE "QUSLSPL ".    (3)
     05  SPC-AUT       PIC  X(10) VALUE "*ALL".
     05  SPC-TEXT      PIC  X(50).
     05  SPC-REPLAC    PIC  X(10) VALUE "*YES".
     05  SPC-DOMAIN    PIC  X(10) VALUE "*USER".
     05  LST-FORMAT-NAME PIC  X(08) VALUE "SPLF0200".  (4)
     05  USR-PRF       PIC  X(10) VALUE "*CURRENT  ".
     05  OUTQ          PIC  X(20) VALUE "*ALL".
     05  FORMTYP       PIC  X(10) VALUE "*ALL".
     05  USRDTA        PIC  X(10) VALUE "*ALL".
     05  JOBNAM        PIC  X(26).
 01  KEYS.                                             (7)
     05  KEY1          PIC S9(09) BINARY VALUE 201.    (8)
     05  KEY2          PIC S9(09) BINARY VALUE 216.
     05  KEY3          PIC S9(09) BINARY VALUE 211.
 01  NUMBER-OF-KEYS    PIC S9(09) BINARY VALUE 3.
 01  MISC2.
     05  PAGESA        PIC  X(04).
     05  PAGESN        REDEFINES PAGESA
                       PIC S9(09) BINARY.
*
 LINKAGE SECTION.
*
* String to map User Space offsets into
*
 01  STRING-SPACE      PIC  X(32000).
*
* User Space Generic Header include.  These includes will be
* mapped over a User Space.
*
 COPY QUSGEN OF QSYSINC-QLBLSRC.                       (11)
*
* List Spool Files API include.  These includes will be
* mapped over a User Space.  The include is copied into the
* source so that we can define the variable length portion
* of QUS-LSPL-KEY-INFO.
*
 01  QUS-LSPL-KEY-INFO.
     05  LEN-FIELD-INFO-RETD                PIC S9(00009) BINARY.
     05  KEY-FIELD-FOR-FIELD-RETD           PIC S9(00009) BINARY.
     05  TYPE-OF-DATA                       PIC  X(00001).
     05  RESERV3                            PIC  X(00003).
     05  DATA-LENGTH                        PIC S9(00009) BINARY.
     05  DATA-FIELD                         PIC  X(00100).
*
*                              Varying length
*    05  RESERVED                           PIC  X(00001).
*
*                              Varying length
 01  QUS-SPLF0200.
     05  NUM-FIELDS-RETD                    PIC S9(00009) BINARY.
     05  KEY-INFO.
         09  LEN-FIELD-INFO-RETD            PIC S9(00009) BINARY.
         09  KEY-FIELD-FOR-FIELD-RETD       PIC S9(00009) BINARY.
         09  TYPE-OF-DATA                   PIC  X(00001).
         09  RESERV3                        PIC  X(00003).
         09  DATA-LENGTH                    PIC S9(00009) BINARY.
         09  DATA-FIELD                     PIC  X(00001).
         09  RESERVED                       PIC  X(00001).
*
*                               Varying length
*
*
* Beginning of mainline
```

```
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Open LISTING file
*
     OPEN OUTPUT LISTING.
*
* Set Error Code structure to use exceptions
*
     MOVE 0 TO BYTES-PROVIDED OF QUS-EC.               (1)
*
* Create a User Space for the List generated by QUSLSPL
*
     CALL "QUSCRTUS" USING SPC-NAME, EXT-ATTR, SPC-SIZE,  (2)
                           SPC-INIT, SPC-AUT, SPC-TEXT,
                           SPC-REPLAC, QUS-EC, SPC-DOMAIN
*
* Call QUSLSPL to get all spooled files for *CURRENT user
*
     CALL "QUSLSPL" USING SPC-NAME, LST-FORMAT-NAME, USR-PRF, (3) (4)
                           OUTQ, FORMTYP, USRDTA, QUS-EC,
                           JOBNAM, KEYS, NUMBER-OF-KEYS.  (5) (6)
*
* Get a resolved pointer to the User Space for performance
*
     CALL "QUSPTRUS" USING SPC-NAME, SPCPTR, QUS-EC.      (9)
*
* If valid information was returned
*
     SET ADDRESS OF QUS-GENERIC-HEADER-0100 TO SPCPTR.

     IF STRUCTURE-RELEASE-LEVEL OF QUS-GENERIC-HEADER-0100  (12)
            NOT EQUAL "0100" WRITE LIST-LINE FROM HDRERR,   (13)
                             STOP RUN.

     IF (INFORMATION-STATUS OF QUS-GENERIC-HEADER-0100 = "C"  (14)
        OR INFORMATION-STATUS OF QUS-GENERIC-HEADER-0100 = "P")
        AND NUMBER-LIST-ENTRIES OF QUS-GENERIC-HEADER-0100 > 0  (16)
*
* address current list entry
*

        SET ADDRESS OF STRING-SPACE TO SPCPTR,

        SET ADDRESS OF QUS-SPLF0200 TO
            ADDRESS OF STRING-SPACE((OFFSET-LIST-DATA
            OF QUS-GENERIC-HEADER-0100 + 1):1),     (18)
*
* and process all of the entries
*
        PERFORM PROCES
           NUMBER-LIST-ENTRIES OF QUS-GENERIC-HEADER-0100 TIMES,  (20)

     ELSE
        WRITE LIST-LINE FROM LSTERR.             (15)
     STOP RUN.                                   (28)
****************************************************************
 PROCES.
*
* address the first variable length record for this entry
*
     SET ADDRESS OF QUS-LSPL-KEY-INFO TO ADDRESS OF
          QUS-SPLF0200(5:).
*
* process all variable length records associated with this entry
*
```

```
           PERFORM PROCES2 NUM-FIELDS-RETD TIMES.  (22) (23)

           WRITE LIST-LINE FROM PRTLIN.               (26)
      *
      * after each entry, increment to the next entry
      *
           SET ADDRESS OF STRING-SPACE TO ADDRESS OF QUS-SPLF0200.  (27)

           SET ADDRESS OF QUS-SPLF0200 TO ADDRESS OF STRING-SPACE
               ((SIZE-EACH-ENTRY OF QUS-GENERIC-HEADER-0100 + 1):1).
      *
      * Process each variable length record based on key
      *
       PROCES2.
      *
      * extract spooled file name for report
      *
           IF KEY-FIELD-FOR-FIELD-RETD OF QUS-LSPL-KEY-INFO = 201  (24) (25)
                       MOVE SPACES TO PRTFIL,
                       MOVE DATA-FIELD OF QUS-LSPL-KEY-INFO(
                          1:DATA-LENGTH OF QUS-LSPL-KEY-INFO)
                          TO PRTFIL.
      *
      * extract number of pages for report
      *
           IF KEY-FIELD-FOR-FIELD-RETD OF QUS-LSPL-KEY-INFO = 211  (24) (25)
                       MOVE DATA-FIELD OF QUS-LSPL-KEY-INFO(
                          1:DATA-LENGTH OF QUS-LSPL-KEY-INFO)
                          TO PAGESA,
                       MOVE PAGESN TO PAGES.
      *
      * extract age of spooled file for report
      *
           IF KEY-FIELD-FOR-FIELD-RETD OF QUS-LSPL-KEY-INFO = 216  (24) (25)
                       MOVE SPACES TO OPNDAT,
                       MOVE DATA-FIELD OF QUS-LSPL-KEY-INFO(
                          1:DATA-LENGTH OF QUS-LSPL-KEY-INFO)
                          TO OPNDAT.
      *
      * address next variable length entry
      *
           SET ADDRESS OF STRING-SPACE TO ADDRESS OF QUS-LSPL-KEY-INFO.

           SET ADDRESS OF QUS-LSPL-KEY-INFO TO ADDRESS OF
                       STRING-SPACE(
                       LEN-FIELD-INFO-RETD OF QUS-LSPL-KEY-INFO + 1:1).
```

Related reference:

"Example in OPM RPG: Using keys with the List Spooled Files (QUSLSPL) API" on page 168
This OPM RPG program processes a list of spooled file information that you have specified using keys.

Example in ILE C: Using keys with the List Spooled Files (QUSLSPL) API

This ILE C program processes a list of spooled file information that you have specified using keys.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*****************************************************************/
/*                                                               */
/*   Program:      List Spooled Files for Current User           */
/*                                                               */
/*   Language:     ILE C                                         */
/*                                                               */
/*   Description:  This example shows the steps necessary to     */
/*                 process keyed output from an API              */
/*                                                               */
```

```
/*   APIs Used:    QUSLSPL  - List Spooled Files                  */
/*                 QUSCRTUS - Create User Space                   */
/*                 QUSPTRUS - Retrieve Pointer to User Space      */
/*                                                                */
/******************************************************************/

#include <stdio.h>
#include <string.h>
#include <quslspl.h>     /* QUSLSPL API header                    */
#include <quscrtus.h>    /* QUSCRTUS API header                   */
#include <qusptrus.h>    /* QUSPTRUS API header                   */
#include <qusgen.h>      /* Format Structures for User Space (11) */
#include <qusec.h>       /* Error Code parameter include for APIs */
#include <qliept.h>      /* Entry Point Table include for APIs    */

/******************************************************************/
/* Global variables                                               */
/******************************************************************/
char     spc_name[20] = "SPCNAME   QTEMP     ";
int      spc_size = 2000;
char     spc_init = 0x00;
char    *spcptr, *lstptr, *lstptr2;
int      pages;
struct   keys { int key1;                                  (7)
                int key2;
                int key3;} keys = {201, 211, 216};         (8)
int      number_of_keys = 3;
char     ext_attr[10] = "QUSLSPL   ";
char     spc_aut[10] = "*ALL      ";
char     spc_text[50] = "                                          ";
char     spc_replac[10] = "*YES      ";
char     spc_domain[10] = "*USER     ";
char     format[8] = "SPLF0200";                    (4)
char     usr_prf[10] = "*CURRENT  ";
char     outq[20] = "*ALL                ";
char     formtyp[10] = "*ALL      ";
char     usrdta[10] = "*ALL      ";
char     jobnam[26] = "                           ";
char     prtfil[10];
char     opndat[7];
typedef  struct {
         Qus_LSPL_Key_Info_t Key_Info;
         char                Data_Field[100];
         } var_record_t;
Qus_EC_t error_code;
int      i, j;
char     prtlin[100];
FILE    *record;

main()
{
   /*************************************************************/
   /* Open print file for report                                */
   /*************************************************************/

   if((record = fopen("QPRINT", "wb, lrecl=132, type=record")) == NULL)
     { printf("File QPRINT could not be opened\n");
       exit();
     }
   /*************************************************************/
   /* Set Error Code structure to use exceptions                */
   /*************************************************************/

   error_code.Bytes_Provided = 0;                          (1)

   /*************************************************************/
   /* Create a User Space for the List generated by QUSLSPL     */
```

```
/*************************************************************/

QUSCRTUS(spc_name,        /* User space name and library  (2)  */
         ext_attr,        /* Extended attribute                */
         spc_size,        /* Initial space size                */
         &spc_init,       /* Initialize value for space        */
         spc_aut,         /* Public authorization              */
         spc_text,        /* Text description                  */
         spc_replac,      /* Replace option                    */
         error_code,      /* Error code structure              */
         spc_domain);     /* Domain of space                   */

/*************************************************************/
/* Call QUSLSPL to get all spooled files for *CURRENT user   */
/*************************************************************/

QUSLSPL( spc_name,        /* User space name and library  (3)  */
         format,          /* API format          (4)           */
         usr_prf,         /* User profile                      */
         outq,            /* Output Queue                      */
         formtyp,         /* Form type                         */
         usrdta,          /* User data                         */
         error_code,      /* Error code structure              */
         jobnam,          /* Job name                          */
         keys,            /* Keys to return      (5)           */
         number_of_keys); /* Number of keys      (6)           */

/*************************************************************/
/* Get a resolved pointer to the User Space                  */
/*************************************************************/

QUSPTRUS(spc_name,        /* User space name and library  (9)  */
         &spcptr,         /* Space pointer                     */
         error_code);     /* Error code structure              */

/*************************************************************/
/* If valid information returned                             */
/*************************************************************/

if(memcmp\
  (((Qus_Generic_Header_0100_t *)spcptr)->Structure_Release_Level,  (12)
     "0100", 4) != 0) { printf("Unknown Generic Header");   (13)
                        exit();
                      }

if((((Qus_Generic_Header_0100_t *)spcptr)->Information_Status=='C')\  (14)
     || (((Qus_Generic_Header_0100_t *)spcptr)->Information_Status\
     == 'P'))
     {
      if(((Qus_Generic_Header_0100_t *)spcptr)->Number_List_Entries\  (16)
           > 0)

/*************************************************************/
/* address current list entry                                */
/*************************************************************/

      {
       lstptr = spcptr + (((Qus_Generic_Header_0100_t *)spcptr)\
                ->Offset_List_Data);

/*************************************************************/
/* process all the entries                                   */
/*************************************************************/

       for(i = 0; i < (((Qus_Generic_Header_0100_t *)spcptr)\ (20)
                       ->Number_List_Entries); i++)
         {
```

```
/************************************************************/
/* set lstptr2 to first variable length record for this entry  */
/************************************************************/

          lstptr2 = lstptr + 4;

/************************************************************/
/* process all the variable length records for this entry      */
/************************************************************/

          for(j = 0; j < (((Qus_SPLF0200_t *)lstptr)\  (22) (23)
                           ->Num_Fields_Retd); j++)
            {

/************************************************************/
/* extract spooled file name for report                     */
/************************************************************/

             if((((Qus_LSPL_Key_Info_t *)lstptr2)\  (24) (25)
                   ->Key_Field_for_Field_Retd) == 201)
                   { memcpy(prtfil, "          ", 10);
                     memcpy(prtfil, (((var_record_t *)\
                           lstptr2)->Data_Field),
                           (((Qus_LSPL_Key_Info_t *)lstptr2)\
                              ->Data_Length));
                   }

/************************************************************/
/* extract number of pages for report                       */
/************************************************************/

             if((((Qus_LSPL_Key_Info_t *)lstptr2)\  (24) (25)
                   ->Key_Field_for_Field_Retd) == 211)
                   { memcpy(&pages, (((var_record_t *)\
                           lstptr2)->Data_Field),
                           (((Qus_LSPL_Key_Info_t *)lstptr2)\
                              ->Data_Length));
                   }

/************************************************************/
/* extract age of spooled file for report                   */
/************************************************************/

             if((((Qus_LSPL_Key_Info_t *)lstptr2)\  (24) (25)
                   ->Key_Field_for_Field_Retd) == 216)
                   { memcpy(opndat, "       ", 7);
                     memcpy(opndat, (((var_record_t *)\
                           lstptr2)->Data_Field),
                           (((Qus_LSPL_Key_Info_t *)lstptr2)\
                              ->Data_Length));
                   }

/************************************************************/
/* bump lstptr2 to next variable length record              */
/************************************************************/

             lstptr2 = lstptr2 +
                           (((Qus_LSPL_Key_Info_t *)lstptr2)\
                              ->Len_Field_Info_Retd);
            }

/************************************************************/
/* print collected information                              */
/************************************************************/

          sprintf(prtlin, "%.10s     %.10d     %.7s",  (26)
```

```
                prtfil, pages, opndat);
             fwrite(prtlin, 1, 100, record);

   /*************************************************************/
   /* bump lstptr to next list entry                            */
   /*************************************************************/

             lstptr += (((Qus_Generic_Header_0100_t *)spcptr)\  (27)
                           ->Size_Each_Entry);
            }

   /*************************************************************/
   /* exit at end of list                                       */
   /*************************************************************/

           fclose(record);
           exit();
          }
         }
        else
         { printf("List data not valid");         (15)
           exit();
         }
}                                                  (28)
```

Related reference:

"Example in OPM RPG: Using keys with the List Spooled Files (QUSLSPL) API" on page 168
This OPM RPG program processes a list of spooled file information that you have specified using keys.

Example in ILE RPG: Using keys with the List Spooled Files (QUSLSPL) API

This ILE RPG program processes a list of spooled file information that you have specified using keys.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
     F*****************************************************************
     F*****************************************************************
     F*
     F*  Program:      List Spooled Files for Current User
     F*
     F*  Language:     ILE RPG
     F*
     F*  Description:  This example shows the steps necessary to
     F*                process keyed output from an API.
     F*
     F*  APIs Used:    QUSLSPL  - List Spooled Files
     F*                QUSCRTUS - Create User Space
     F*                QUSPTRUS - Retrieve Pointer to User Space
     F*
     F*****************************************************************
     F*****************************************************************
     F*
     FQPRINT    O    F  132        PRINTER OFLIND(*INOF)
     D*
     D* Error Code parameter include
     D*
     D/COPY QSYSINC/QRPGLESRC,QUSEC                        (11)
     D*
     DSPC_NAME         S             20    INZ('SPCNAME   QTEMP     ')
     DSPC_SIZE         S              9B 0 INZ(2000)        (2)
     DSPC_INIT         S              1    INZ(X'00')
     DLSTPTR           S               *
     DLSTPTR2          S               *
     DSPCPTR           S               *
     DARR              S              1    BASED(LSTPTR) DIM(32767)
     D                 DS
```

```
DPAGES#                   1      4B 0
DPAGESA                   1      4
DKEYS             DS                                  (7)
D                                9B 0 INZ(201)        (8)
D                                9B 0 INZ(216)
D                                9B 0 INZ(211)
DKEY#             S              9B 0 INZ(3)
D*****************************************************************
D*
D* The following QUSGEN include from QSYSINC is copied into   (11)
D* this program so that it can be declared as BASED on SPCPTR
D*
D*****************************************************************
DQUSH0100         DS                  BASED(SPCPTR)
D*                                               Qus Generic Header 0100
D QUSUA                   1     64
D*                                               User Area
D QUSSGH                 65     68B 0
D*                                               Size Generic Header
D QUSSRL                 69     72
D*                                               Structure Release Level
D QUSFN                  73     80
D*                                               Format Name
D QUSAU                  81     90
D*                                               API Used
D QUSDTC                 91    103
D*                                               Date Time Created
D QUSIS                 104    104
D*                                               Information Status
D QUSSUS                105    108B 0
D*                                               Size User Space
D QUSOIP                109    112B 0
D*                                               Offset Input Parameter
D QUSSIP                113    116B 0
D*                                               Size Input Parameter
D QUSOHS                117    120B 0
D*                                               Offset Header Section
D QUSSHS                121    124B 0
D*                                               Size Header Section
D QUSOLD                125    128B 0
D*                                               Offset List Data
D QUSSLD                129    132B 0
D*                                               Size List Data
D QUSNBRLE              133    136B 0
D*                                               Number List Entries
D QUSSEE                137    140B 0
D*                                               Size Each Entry
D QUSSIDLE              141    144B 0
D*                                               CCSID List Ent
D QUSCID                145    146
D*                                               Country ID
D QUSLID                147    149
D*                                               Language ID
D QUSSLI                150    150
D*                                               Subset List Indicator
D QUSERVED00            151    192
D*                                               Reserved
D*****************************************************************
D*
D* The following QUSLSPL include from QSYSINC is copied into
D* this program so that it can be declared as BASED
D*
D*****************************************************************
D*****************************************************************
D*Prototype for calling List Spooled File API QUSLSPL
D*****************************************************************
D QUSLSPL         C                   'QUSLSPL'
```

```
D*****************************************************************
D*Type definition for the SPLF0200 format.
D*******
D*NOTE:  The following type definition only defines the fixed
D*   portion of the format. Any varying length field will
D*   have to be defined by the user.
D*****************************************************************
DQUSSPLKI         DS           100    BASED(LSTPTR2)
D*                                             Qus LSPL Key Info
D QUSLFIR02               1      4B 0
D*                                             Len Field Info Retd
D QUSKFFFR00              5      8B 0
D*                                             Key Field for Field Retd
D QUSTOD02                9      9
D*                                             Type of Data
D QUSR300                10     12
D*                                             Reserv3
D QUSDL02                13     16B 0
D*                                             Data Length
D*QUSDATA08              17     17
D*
D*                             Varying length
D*QUSERVED34             18     18
D*
D*                             Varying length
DQUSF0200         DS                  BASED(LSTPTR)
D*                                             Qus SPLF0200
D QUSNBRFR00              1      4B 0
D*                                             Num Fields Retd
D*QUSKI00                 18
D* QUSLFIR03               5      8B 0
D* QUSKFFFR01              9     12B 0
D* QUSTOD03               13     13
D* QUSR301                14     16
D* QUSDL03                17     20B 0
D* QUSDATA09              21     21
D* QUSERVED35             22     22
D*
D*                              Varying length
C*
C* Start of mainline
C*
C*
C* Set Error Code structure to use exceptions
C*
C                   Z-ADD     0             QUSBPRV    (1)
C*
C* Create a User Space for the List generated by QUSLSPL
C*
C                   CALL      'QUSCRTUS'                (2)
C                   PARM                    SPC_NAME
C                   PARM      'QUSLSPL   '  EXT_ATTR         10
C                   PARM                    SPC_SIZE
C                   PARM                    SPC_INIT
C                   PARM      '*ALL'        SPC_AUT          10
C                   PARM      *BLANKS       SPC_TEXT         50
C                   PARM      '*YES'        SPC_REPLAC       10
C                   PARM                    QUSEC
C                   PARM      '*USER'       SPC_DOMAIN       10
C*
C* Call QUSLSPL to get all spooled files for *CURRENT user
C*
C                   CALL      'QUSLSPL'                 (3)
C                   PARM                    SPC_NAME
C                   PARM      'SPLF0200'    FORMAT            8  (4)
C                   PARM      '*CURRENT'    USR_PRF          10
C                   PARM      '*ALL'        OUTQ             20
```

```
C                   PARM      '*ALL'        FORMTYP          10
C                   PARM      '*ALL'        USRDTA           10
C                   PARM                    QUSEC
C                   PARM                    JOBNAM           26
C                   PARM                    KEYS        (5)
C                   PARM                    KEY#        (6)
C*
C* Get a resolved pointer to the User Space for performance
C*
C                   CALL      'QUSPTRUS'                (9)
C                   PARM                    SPC_NAME
C                   PARM                    SPCPTR
C                   PARM                    QUSEC
C*
C* If valid information was returned
C*
C     QUSSRL        IFEQ      '0100'                    (12)
C     QUSIS         IFEQ      'C'                       (14)
C     QUSIS         OREQ      'P'
C*
C* and list entries were found
C*
C     QUSNBRLE      IFGT      0                         (16)
C*
C* set LSTPTR to the first byte of the User Space
C*
C                   EVAL      LSTPTR = SPCPTR
C*
C* increment LSTPTR to the first List entry
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSOLD + 1))  (18)
C*
C* and process all of the entries
C*
C                   DO        QUSNBRLE                  (20)
C*
C* set LSTPTR2 to the first variable length record for this entry
C*
C                   Z-ADD     5             X                9 0
C                   EVAL      LSTPTR2 = %ADDR(ARR(X))   (22)
C                   DO        QUSNBRFR00                (23)
C*
C* process the data based on key type
C*
C     QUSKFFFR00    CASEQ     201           FILNAM      (24)
C     QUSKFFFR00    CASEQ     211           PAGES
C     QUSKFFFR00    CASEQ     216           AGE
C                   CAS                     ERROR
C                   END
C*
C* increment LSTPTR2 to next variable length record
C*
C                   ADD       QUSLFIR02     X
C                   EVAL      LSTPTR2 = %ADDR(ARR(X))
C                   END
C                   EXCEPT    PRTLIN                    (26)
C*
C* after each entry, increment LSTPTR to the next entry
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSSEE + 1))  (27)
C                   END
C                   END
C                   ELSE
C                   EXCEPT    LSTERR                    (15)
C                   END
C                   ELSE
C                   EXCEPT    HDRERR                    (13)
```

```
C                   END
C*
C* Exit the program
C*
C                   EVAL      *INLR = '1'              (28)
C                   RETURN
C*****************************************************************
C     FILNAM        BEGSR
C*
C* extract spooled file name for report
C*
C                   MOVE      *BLANKS       PRTFIL           10
C                   EVAL      PRTFIL = %SUBST(QUSSPLKI:17:QUSDL02)  (25)
C                   ENDSR
C*****************************************************************
C     PAGES         BEGSR
C*
C* extract number of pages for report
C*
C                   EVAL      PAGESA = %SUBST(QUSSPLKI:17:QUSDL02)  (25)
C                   ENDSR
C*****************************************************************
C     AGE           BEGSR
C*
C* extract age of spooled file for report
C*
C                   MOVE      *BLANKS       OPNDAT            7
C                   EVAL      OPNDAT = %SUBST(QUSSPLKI:17:QUSDL02)  (25)
C                   ENDSR
C*****************************************************************
C     ERROR         BEGSR
C     QUSKFFFR00    DSPLY
C                   EVAL      *INLR = '1'
C                   RETURN
C                   ENDSR
C*****************************************************************
OQPRINT    E            PRTLIN         1
O                       PRTFIL              10
O                       PAGES#              25
O                       OPNDAT              40
OQPRINT    E            LSTERR         1
O                                           22 'List data not valid'
OQPRINT    E            HDRERR         1
O                                           22 'Unknown Generic Header'
```

Related reference:

"Example in OPM RPG: Using keys with the List Spooled Files (QUSLSPL) API" on page 168
This OPM RPG program processes a list of spooled file information that you have specified using keys.

Examples: Service-program-based APIs

These program examples demonstrate the use of service-program-based APIs in several different high-level language programs. The example APIs represent two general functions of APIs: change and retrieve.

The examples use the registration facility APIs. The registration facility APIs provide a means for storing and retrieving information about exit points and exit programs. An *exit point* is a specific point in a system function or program where control is passed to one or more exit programs. An *exit program* is a program to which control is passed from an exit point. The examples show how to manipulate exit points and exit programs, how to retrieve information about exit points and exit programs that are stored with the registration facility, and how to call an exit program.

Several of the registration facility APIs manipulate the information that the registration facility repository contains. One API is provided for retrieving information from the repository.

For a detailed description of how to use the API, see "API information format" on page 47. These descriptions and the programs that support them are in RPG. You can, however, view the same programs in different languages.

Related concepts:

"APIs for the service-program-based environment" on page 10
APIs based on service programs are called as procedures exported from ILE service programs (*SRVPGM).

Related reference:

Generic header files using ILE APIs

Example: Keyed interface using ILE APIs

Error handling using ILE APIs

Examples: Receiver variables using ILE APIs

Example in ILE C: Registering exit points and adding exit programs

This ILE C program registers an exit point with the registration facility. After the successful completion of the registration, the program adds an exit program to the exit point.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*  PROGRAM:      Register an Exit Point                             */
/*                Add an Exit Program                                */
/*                                                                   */
/*  LANGUAGE:     ILE C                                              */
/*                                                                   */
/*  DESCRIPTION:  This program registers an exit point with the      */
/*                registration facility.  After the successful       */
/*                completion of the registration of the exit point,  */
/*                an exit program is added to the exit point.        */
/*                                                                   */
/*  APIs USED:    QusRegisterExitPoint - Register Exit Point         */
/*                QusAddExitProgram    - Add Exit Program            */
/*                                                                   */
/*********************************************************************/
/*  NOTE:  This example uses APIs that are shipped with *EXCLUDE     */
/*         authority.  The user needs *USE authority to the service  */
/*         program QUSRGFA1 to use these APIs.                       */
/*********************************************************************/

/*********************************************************************/
/*                     Includes                                      */
/*********************************************************************/
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <qusrgfa1.h>
#include <qusec.h>
#include <qliept.h>

/*********************************************************************/
/*                     Structures                                    */
/*********************************************************************/

typedef struct {                     /* Error code                */
 Qus_EC_t ec_fields;
 char     exception_data[100];
} error_code_struct;

typedef struct {                     /* Exit point control keys    */
  int              num_rec;
```

```
  Qus_Vlen_Rec_4_t max_pgms_rec;
  int              max_pgms;
  Qus_Vlen_Rec_4_t descrip_rec;
  char             text_desc[50];
} rgpt_controls;

typedef struct {                     /* Exit program attribute keys*/
  int              num_rec;
  Qus_Vlen_Rec_4_t replace_rec;
  char             replace;
  char             Reserved[3];
  Qus_Vlen_Rec_4_t CCSID_rec;
  int              CCSID;
} addep_attributes;

/*********************************************************************/
/*                                                                   */
/*                            main                                   */
/*                                                                   */
/*********************************************************************/
int main()
{
  int  ccsid,
       pgm_num,
       num_of_attrs,
       epgm_num,
       len_epgm_data,
       add_epgm_num,
       *ccsid_ptr,
       *pgm_num_ptr;
  error_code_struct error_code;
  rgpt_controls control_keys;
  addep_attributes attrib_keys;

  /*******************************************************************/
  /* Register the exit point with the registration facility. If the */
  /* registration of the exit point is successful, add an exit      */
  /* program to the exit point.                                     */
  /*******************************************************************/

  /*******************************************************************/
  /* Initialize the error code parameter.  To signal exceptions to  */
  /* this program by the API, you need to set the bytes provided    */
  /* field of the error code to zero.  Because this program has     */
  /* exceptions sent back through the error code parameter, it sets */
  /* the bytes provided field to the number of bytes that it gives  */
  /* the API for the parameter.                                     */
  /*******************************************************************/
  error_code.ec_fields.Bytes_Provided=sizeof(error_code_struct);

  /*******************************************************************/
  /* Set the exit point controls. Each control field is passed to   */
  /* the API using a variable length record.  Each record must      */
  /* start on a 4-byte boundary.                                    */
  /*******************************************************************/

  /*******************************************************************/
  /* Set the total number of controls that are being specified on   */
  /* the call.  This program lets the API take the default for the  */
  /* controls that are not specified.                               */
  /*******************************************************************/
  control_keys.num_rec=2;

  /*******************************************************************/
  /* Set the values for the two controls that are specified:        */
  /*   Maximum number of exit programs = 10                         */
  /*   Exit point text description     = "EXIT POINT EXAMPLE"       */
```

```
/*****************************************************************/
control_keys.max_pgms_rec.Length_Vlen_Record=16;
control_keys.max_pgms_rec.Control_Key=3;
control_keys.max_pgms_rec.Length_Data=4;
control_keys.max_pgms=10;

control_keys.descrip_rec.Length_Vlen_Record=62;
control_keys.descrip_rec.Control_Key=8;
control_keys.descrip_rec.Length_Data=50;
memcpy(control_keys.text_desc,
       "EXIT POINT EXAMPLE                              ",50);


/*****************************************************************/
/* Call the API to register the exit point.                      */
/*****************************************************************/
QusRegisterExitPoint("EXAMPLE_EXIT_POINT  ",
                     "EXMP0100",
                     &control_keys,
                     &error_code);

/*****************************************************************/
/* If an exception occurs, the API returns the exception in the  */
/* error code parameter.  The bytes available field is set to    */
/* zero if no exception occurs and nonzero if an exception does  */
/* occur.                                                        */
/*****************************************************************/
if (error_code.ec_fields.Bytes_Available != 0)
{
   printf("ATTEMPT TO REGISTER EXIT POINT FAILED WITH EXCEPTION: %.7s",
           error_code.ec_fields.Exception_Id);
   exit(1);
}

/*****************************************************************/
/* If the call to register an exit point is successful, add      */
/* an exit program to the exit point.                            */
/*****************************************************************/

/*****************************************************************/
/* Set the total number of exit program attributes that are being */
/* specified on the call.  This program lets the API take the    */
/* default for the attributes that are not specified.  Each      */
/* attribute record must be 4-byte aligned.                      */
/*****************************************************************/
attrib_keys.num_rec=2;

/*****************************************************************/
/* Set the values for the two attributes that are being          */
/* specified:                                                    */
/*       Replace exit program    = 1                             */
/*       Exit program data CCSID = 37                            */
/*****************************************************************/
attrib_keys.replace_rec.Length_Vlen_Record=16;
attrib_keys.replace_rec.Control_Key=4;
attrib_keys.replace_rec.Length_Data=1;
attrib_keys.replace='1';

attrib_keys.CCSID_rec.Length_Vlen_Record=16;
attrib_keys.CCSID_rec.Control_Key=3;
attrib_keys.CCSID_rec.Length_Data=4;
attrib_keys.CCSID=37;

/*****************************************************************/
/* Call the API to add the exit program.                         */
/*****************************************************************/
QusAddExitProgram("EXAMPLE_EXIT_POINT  ",
```

```
                  "EXMP0100",
                  1,
                  "EXAMPLEPGMEXAMPLELIB",
                  "EXAMPLE EXIT PROGRAM DATA",
                  25,
                  &attrib_keys,
                  &error_code);

  /*************************************************************/
  /* If an exception occurs, the API returns the exception in the   */
  /* error code parameter.  The bytes available field is set to     */
  /* zero if no exception occurs and nonzero if an exception does   */
  /* occur.                                                         */
  /*************************************************************/
  if (error_code.ec_fields.Bytes_Available != 0)
  {
     printf("ATTEMPT TO ADD AN EXIT PROGRAM FAILED WITH EXCEPTION: %.7s",
             error_code.ec_fields.Exception_Id);
     exit(1);
  }


} /* End program */
```

Example in OPM COBOL: Registering exit points and adding exit programs

This OPM COBOL program registers an exit point with the registration facility. After the successful completion of the registration, the program adds an exit program to the exit point.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
       IDENTIFICATION DIVISION.
      *************************************************************
      *************************************************************
      *
      *  Program:      Register an Exit Point
      *                Add an Exit Program
      *
      *  Language:     OPM COBOL
      *
      *  Description:  This program registers an exit point with the
      *                registration facility.  After the successful
      *                completion of the registration of the exit point,
      *                an exit program is added to the exit point.
      *
      *  APIs Used:    QUSRGPT  - Register Exit Point
      *                QUSADDEP - Add Exit Program
      *
      *************************************************************
      *
      *************************************************************
       PROGRAM-ID. REGFAC1.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT LISTING ASSIGN TO PRINTER-QPRINT
                          ORGANIZATION IS SEQUENTIAL.
       DATA DIVISION.
       FILE SECTION.
       FD  LISTING RECORD CONTAINS 132 CHARACTERS
                   LABEL RECORDS ARE STANDARD
                   DATA RECORD IS LIST-LINE.
       01  LIST-LINE        PIC X(132).
       WORKING-STORAGE SECTION.
      *
      * Keyed Variable Length Record includes
      *
```

```
 COPY QUS OF QSYSINC-QLBLSRC.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Error message text
*
 01  BAD-REG.
     05  TEXT1        PIC X(39)
                 VALUE "Attempt to register exit point failed: ".
     05  EXCEPTION-ID PIC X(07).
 01  BAD-ADD.
     05  TEXT1        PIC X(36)
               VALUE "Attempt to add exit program failed: ".
     05  EXCEPTION-ID PIC X(07).
*
* Miscellaneous elements
*
 01  VARREC.
     05  NBR-RECORDS PIC S9(09) BINARY.
     05  VAR-RECORDS PIC  X(1000).
 01  MISC.
     05  VAR-OFFSET      PIC S9(09) VALUE 1.
     05  BINARY-NUMBER   PIC S9(09) BINARY.
     05  BINARY-CHAR REDEFINES BINARY-NUMBER PIC X(04).
     05  X               PIC S9(09) BINARY.
     05  EXIT-POINT-NAME PIC  X(20) VALUE "EXAMPLE_EXIT_POINT".
     05  EXIT-PGM        PIC  X(20) VALUE "EXAMPLEPGMEXAMPLELIB".
     05  EXIT-PGM-NBR    PIC S9(09) VALUE 1 BINARY.
     05  EXIT-PGM-DATA   PIC  X(25)
                            VALUE "EXAMPLE EXIT PROGRAM DATA".
     05  FORMAT-NAME     PIC  X(08) VALUE "EXMP0100".
*
* Beginning of mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Register the exit point with the registration facility.  If the
* registration of the exit point is successful, add an exit
* program to the exit point.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
* API for the parameter.
*
     MOVE 16 TO BYTES-PROVIDED.
*
* Set the exit point controls.  Each control field is passed to
* the API using a variable length record.  Each record must
* start on a 4-byte boundary.
*
* Set the total number of controls that are being specified on
* the call.  This program lets the API take the default for the
* controls that are not specified.
*
     MOVE 2 TO NBR-RECORDS.
```

```
 *
 * Set the values for the two controls that are specified:
 *   Maximum number of exit programs = 10
 *   Exit point description = 'EXIT POINT EXAMPLE'
 *
      MOVE 3 TO CONTROL-KEY OF QUS-VLEN-REC-4.
      MOVE 4 TO LENGTH-DATA OF QUS-VLEN-REC-4.
      MOVE 10 TO BINARY-NUMBER.
      MOVE BINARY-CHAR TO VAR-RECORDS((VAR-OFFSET + 12):4).
      PERFORM CALCULATE-NEXT-OFFSET.
      MOVE 8 TO CONTROL-KEY OF QUS-VLEN-REC-4.
      MOVE 50 TO LENGTH-DATA OF QUS-VLEN-REC-4.
      MOVE "EXIT POINT EXAMPLE"
            TO VAR-RECORDS((VAR-OFFSET + 12):50).
      PERFORM CALCULATE-NEXT-OFFSET.
C*
C* Call the API to add the exit point.
C*
      CALL "QUSRGPT" USING EXIT-POINT-NAME OF MISC,
                           FORMAT-NAME OF MISC,
                           VARREC, QUS-EC.
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
      IF BYTES-AVAILABLE OF QUS-EC > 0
                            OPEN OUTPUT LISTING,
                            MOVE EXCEPTION-ID OF QUS-EC
                                 TO EXCEPTION-ID OF BAD-REG,
                            WRITE LIST-LINE FROM BAD-REG,
                            STOP RUN.
 *
 * If the call to register an exit point is successful, add
 * an exit program to the exit point.
 *
 * Set the total number of exit program attributes that are being
 * specified on the call.  This program lets the API take the
 * default for the attributes that are not specified.  Each
 * attribute record must be 4-byte aligned.
 *
      MOVE 2 TO NBR-RECORDS.
      MOVE 1 TO VAR-OFFSET.
 *
 * Set the values for the two attributes that are being specified:
 *   Replace exit program    = 1
 *   Exit program data CCSID = 37
 *
      MOVE 4 TO CONTROL-KEY OF QUS-VLEN-REC-4.
      MOVE 1 TO LENGTH-DATA OF QUS-VLEN-REC-4.
      MOVE 1 TO VAR-RECORDS((VAR-OFFSET + 12):1).
      PERFORM CALCULATE-NEXT-OFFSET.
      MOVE 3 TO CONTROL-KEY OF QUS-VLEN-REC-4.
      MOVE 4 TO LENGTH-DATA OF QUS-VLEN-REC-4.
      MOVE 37 TO BINARY-NUMBER.
      MOVE BINARY-CHAR TO VAR-RECORDS((VAR-OFFSET + 12):4).
      PERFORM CALCULATE-NEXT-OFFSET.
 *
 * Call the API to register the exit program.
 *
      CALL "QUSADDEP" USING EXIT-POINT-NAME OF MISC,
                            FORMAT-NAME OF MISC,
                            EXIT-PGM-NBR OF MISC,
                            EXIT-PGM OF MISC,
                            EXIT-PGM-DATA OF MISC,
                       BY CONTENT LENGTH OF EXIT-PGM-DATA OF MISC,
```

```
                          VARREC, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           OPEN OUTPUT LISTING,
                           MOVE EXCEPTION-ID OF QUS-EC
                                TO EXCEPTION-ID OF BAD-ADD,
                           WRITE LIST-LINE FROM BAD-ADD,
                           STOP RUN.
*
     STOP RUN.
*
* End of MAINLINE
*
*
* Calculate 4-byte aligned offset for next variable length record
*
 CALCULATE-NEXT-OFFSET.
     COMPUTE BINARY-NUMBER = LENGTH-DATA OF QUS-VLEN-REC-4 + 12.
     DIVIDE BINARY-NUMBER BY 4 GIVING BINARY-NUMBER REMAINDER X.
     IF X = 0 COMPUTE LENGTH-VLEN-RECORD OF QUS-VLEN-REC-4 =
                      LENGTH-DATA OF QUS-VLEN-REC-4 + 12
        ELSE  COMPUTE LENGTH-VLEN-RECORD OF QUS-VLEN-REC-4 =
                      LENGTH-DATA OF QUS-VLEN-REC-4 + 12 +
                      ( 4 - X ).
     MOVE QUS-VLEN-REC-4 TO VAR-RECORDS(VAR-OFFSET:12).
     COMPUTE VAR-OFFSET = VAR-OFFSET + LENGTH-VLEN-RECORD OF
                          QUS-VLEN-REC-4.
```

Example in ILE COBOL: Registering exit points and adding exit programs

This ILE COBOL program registers an exit point with the registration facility. After the successful completion of the registration, the program adds an exit program to the exit point.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*  Program:      Register an Exit Point
*                Add an Exit Program
*
*  Language:     ILE COBOL
*
*  Description:  This program registers an exit point with the
*                registration facility.  After the successful
*                completion of the registration of the exit point,
*                an exit program is added to the exit point.
*
*  APIs Used:    QusRegisterExitPoint  - Register Exit Point
*                QusAddExitProgram - Add Exit Program
*
*****************************************************************
*
*****************************************************************
 PROGRAM-ID. REGFAC1.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
 DATA DIVISION.
```

```
 FILE SECTION.
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
 WORKING-STORAGE SECTION.
*
* Keyed Variable Length Record includes
*
 COPY QUS OF QSYSINC-QLBLSRC.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Error message text
*
 01  BAD-REG.
     05  TEXT1        PIC X(39)
                 VALUE "Attempt to register exit point failed: ".
     05  EXCEPTION-ID PIC X(07).
 01  BAD-ADD.
     05  TEXT1        PIC X(36)
               VALUE "Attempt to add exit program failed: ".
     05  EXCEPTION-ID PIC X(07).
*
* Miscellaneous elements
*
 01  VARREC.
     05  NBR-RECORDS PIC S9(09) BINARY.
     05  VAR-RECORDS PIC  X(1000).
 01  MISC.
     05  VAR-OFFSET      PIC S9(09) VALUE 1.
     05  BINARY-NUMBER   PIC S9(09) BINARY.
     05  BINARY-CHAR REDEFINES BINARY-NUMBER PIC X(04).
     05  X               PIC S9(09) BINARY.
     05  EXIT-POINT-NAME PIC  X(20) VALUE "EXAMPLE_EXIT_POINT".
     05  EXIT-PGM        PIC  X(20) VALUE "EXAMPLEPGMEXAMPLELIB".
     05  EXIT-PGM-NBR    PIC S9(09) VALUE 1 BINARY.
     05  EXIT-PGM-DATA   PIC  X(25)
                            VALUE "EXAMPLE EXIT PROGRAM DATA".
     05  FORMAT-NAME     PIC  X(08) VALUE "EXMP0100".
*
* Beginning of mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Register the exit point with the registration facility.  If the
* registration of the exit point is successful, add an exit
* program to the exit point.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
* API for the parameter.
*
     MOVE 16 TO BYTES-PROVIDED.
*
```

```
* Set the exit point controls.  Each control field is passed to
* the API using a variable length record.  Each record must
* start on a 4-byte boundary.
*
* Set the total number of controls that are being specified on
* the call.  This program lets the API take the default for the
* controls that are not specified.
*
     MOVE 2 TO NBR-RECORDS.
*
* Set the values for the two controls that are specified:
*   Maximum number of exit programs = 10
*   Exit point description = 'EXIT POINT EXAMPLE'
*
     MOVE 3 TO CONTROL-KEY OF QUS-VLEN-REC-4.
     MOVE 4 TO LENGTH-DATA OF QUS-VLEN-REC-4.
     MOVE 10 TO BINARY-NUMBER.
     MOVE BINARY-CHAR TO VAR-RECORDS((VAR-OFFSET + 12):4).
     PERFORM CALCULATE-NEXT-OFFSET.
     MOVE 8 TO CONTROL-KEY OF QUS-VLEN-REC-4.
     MOVE 50 TO LENGTH-DATA OF QUS-VLEN-REC-4.
     MOVE "EXIT POINT EXAMPLE"
           TO VAR-RECORDS((VAR-OFFSET + 12):50).
     PERFORM CALCULATE-NEXT-OFFSET.
*
* Call the API to add the exit point.
*
     CALL PROCEDURE "QusRegisterExitPoint" USING
                           EXIT-POINT-NAME OF MISC,
                           FORMAT-NAME OF MISC,
                           VARREC, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                             OPEN OUTPUT LISTING,
                             MOVE EXCEPTION-ID OF QUS-EC
                                  TO EXCEPTION-ID OF BAD-REG,
                             WRITE LIST-LINE FROM BAD-REG,
                             STOP RUN.
*
* If the call to register an exit point is successful, add
* an exit program to the exit point.
*
* Set the total number of exit program attributes that are being
* specified on the call.  This program lets the API take the
* default for the attributes that are not specified.  Each
* attribute record must be 4-byte aligned.
*
     MOVE 2 TO NBR-RECORDS.
     MOVE 1 TO VAR-OFFSET.
*
* Set the values for the two attributes that are being specified:
*   Replace exit program    = 1
*   Exit program data CCSID = 37
*
     MOVE 4 TO CONTROL-KEY OF QUS-VLEN-REC-4.
     MOVE 1 TO LENGTH-DATA OF QUS-VLEN-REC-4.
     MOVE 1 TO VAR-RECORDS((VAR-OFFSET + 12):1).
     PERFORM CALCULATE-NEXT-OFFSET.
     MOVE 3 TO CONTROL-KEY OF QUS-VLEN-REC-4.
     MOVE 4 TO LENGTH-DATA OF QUS-VLEN-REC-4.
     MOVE 37 TO BINARY-NUMBER.
     MOVE BINARY-CHAR TO VAR-RECORDS((VAR-OFFSET + 12):4).
```

```
     PERFORM CALCULATE-NEXT-OFFSET.
 *
 * Call the API to register the exit program.
 *
     CALL PROCEDURE "QusAddExitProgram" USING
                      EXIT-POINT-NAME OF MISC,
                      FORMAT-NAME OF MISC,
                      EXIT-PGM-NBR OF MISC,
                      EXIT-PGM OF MISC,
                      EXIT-PGM-DATA OF MISC,
                      BY CONTENT LENGTH OF EXIT-PGM-DATA OF MISC,
                      VARREC, QUS-EC.
 *
 * If an exception occurs, the API returns the exception in the
 * error code parameter.  The bytes available field is set to
 * zero if no exception occurs and greater than zero if an
 * exception does occur.
 *
     IF BYTES-AVAILABLE OF QUS-EC > 0
                             OPEN OUTPUT LISTING,
                             MOVE EXCEPTION-ID OF QUS-EC
                                  TO EXCEPTION-ID OF BAD-ADD,
                             WRITE LIST-LINE FROM BAD-ADD,
                             STOP RUN.
 *
     STOP RUN.
 *
 * End of MAINLINE
 *
 *
 * Calculate 4-byte aligned offset for next variable length record
 *
  CALCULATE-NEXT-OFFSET.
     COMPUTE BINARY-NUMBER = LENGTH-DATA OF QUS-VLEN-REC-4 + 12.
     DIVIDE BINARY-NUMBER BY 4 GIVING BINARY-NUMBER REMAINDER X.
     IF X = 0 COMPUTE LENGTH-VLEN-RECORD OF QUS-VLEN-REC-4 =
                      LENGTH-DATA OF QUS-VLEN-REC-4 + 12
        ELSE  COMPUTE LENGTH-VLEN-RECORD OF QUS-VLEN-REC-4 =
                      LENGTH-DATA OF QUS-VLEN-REC-4 + 12 +
                      ( 4 - X ).
     MOVE QUS-VLEN-REC-4 TO VAR-RECORDS(VAR-OFFSET:12).
     COMPUTE VAR-OFFSET = VAR-OFFSET + LENGTH-VLEN-RECORD OF
                          QUS-VLEN-REC-4.
```

Example in OPM RPG: Registering exit points and adding exit programs

This OPM RPG program registers an exit point with the registration facility. After the successful completion of the registration, the program adds an exit program to the exit point.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*************************************************************
F*************************************************************
F*
F*  Program:      Register an Exit Point
F*                Add an Exit Program
F*
F*  Language:     OPM RPG
F*
F*  Description:  This program registers an exit point with the
F*                registration facility.  After the successful
F*                completion of the registration of the exit point,
F*                an exit program is added to the exit point.
F*
F*  APIs Used:    QUSRGPT - Register Exit Point
F*                QUSADDEP - Add Exit Program
```

```
F*
F*****************************************************************
F*****************************************************************
F*
FQPRINT  O   F     132            PRINTER                        UC
E* COMPILE TIME ARRAY
E                    REC      1000  1
I*
I* Keyed Variable Length Record includes
I*
I/COPY QSYSINC/QRPGSRC,QUS
I*
I* Error Code parameter include.  As this sample program
I* uses /COPY to include the error code structure, only the first
I* 16 bytes of the error code structure are available.  If the
I* application program needs to access the variable length
I* exception data for the error, the developer should physically
I* copy the QSYSINC include and modify the copied include to
I* define additional storage for the exception data.
I*
I/COPY QSYSINC/QRPGSRC,QUSEC
I*
I* Miscellaneous data
I*
IVARREC      DS                          1008
I                                    B   1   40NBRREC
I                                        51004 REC
I I            1                     B100510080VO
I*
IOVRLAY      DS
I                                    B   1   40BINARY
I                                        1   4 BINC
I*
I            DS
I I            'EXAMPLE_EXIT_POINT  '    1  20 EPNTNM
I I            'EXAMPLEPGMEXAMPLELIB'   21  40 EPGM
I I            'EXAMPLE EXIT PROGRAM-   41  65 EPGMDT
I              ' DATA'
I I            'EXAMPLE POINT EXAMPL-   66 115 EPTXT
I              'E'
I I            25                    B 116 1190EPGMSZ
C*
C* Beginning of mainline
C*
C* Register the exit point with the registration facility.  If the
C* registration of the exit point is successful, add an exit
C* program to the exit point.
C*
C* Initialize the error code parameter.  To signal exceptions to
C* this program by the API, you need to set the bytes provided
C* field of the error code to zero.  Because this program has
C* exceptions sent back through the error code parameter, it sets
C* the bytes provided field to the number of bytes it gives the
C* API for the parameter.
C*
C                     Z-ADD16        QUSBNB
C*
C* Set the exit point controls.  Each control field is passed to
C* the API using a variable length record.  Each record must
C* start on a 4-byte boundary.
C*
C* Set the total number of controls that are being specified on
C* the call.  This program lets the API take the default for the
C* controls that are not specified.
C*
C                     Z-ADD2         NBRREC
C*
```

```
C* Set the values for the two controls that are specified:
C*   Maximum number of exit programs = 10
C*   Exit point description = 'EXIT POINT EXAMPLE'
C*
C                     Z-ADD3         QUSBCC
C                     Z-ADD4         QUSBCD
C                     Z-ADD10        BINARY
C           12        ADD  VO        OF      50
C                     MOVEABINC      REC,OF
C                     EXSR CALCVO
C                     Z-ADD8         QUSBCC
C                     Z-ADD50        QUSBCD
C           12        ADD  VO        OF      50
C                     MOVEAEPTXT     REC,OF
C                     EXSR CALCVO
C*
C* Call the API to register the exit point.
C*
C                     CALL 'QUSRGPT'
C                     PARM           EPNTNM
C                     PARM 'EXMP0100'FORMAT  8
C                     PARM           VARREC
C                     PARM           QUSBN
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C           QUSBNC    IFGT 0
C                     OPEN QPRINT
C                     EXCPTERREPT
C                     EXSR DONE
C                     ENDIF
C*
C* If the call to register an exit point is successful, add
C* an exit program to the exit point.
C*
C* Set the total number of exit program attributes that are being
C* specified on the call.  This program lets the API take the
C* default for the attributes that are not specified.  Each
C* attribute record must be 4-byte aligned.
C*
C                     Z-ADD2         NBRREC
C                     Z-ADD1         VO
C*
C* Set the values for the two attributes that are being specified:
C*   Replace exit program    = 1
C*   Exit program data CCSID = 37
C*
C                     Z-ADD4         QUSBCC
C                     Z-ADD1         QUSBCD
C           12        ADD  VO        OF      50
C                     MOVE '1'       REC,OF
C                     EXSR CALCVO
C                     Z-ADD3         QUSBCC
C                     Z-ADD4         QUSBCD
C                     Z-ADD37        BINARY
C           12        ADD  VO        OF      50
C                     MOVEABINC      REC,OF
C                     EXSR CALCVO
C*
C* Call the API to add the exit program.
C*
C                     CALL 'QUSADDEP'
C                     PARM           EPNTNM
C                     PARM 'EXMP0100'FORMAT
```

```
C                     PARM 1         BINARY
C                     PARM           EPGM
C                     PARM           EPGMDT
C                     PARM           EPGMSZ
C                     PARM           VARREC
C                     PARM           QUSBN
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C           QUSBNC    IFGT 0
C                     OPEN QPRINT
C                     EXCPTERRPGM
C                     EXSR DONE
C                     ENDIF
C                     EXSR DONE
C*
C* End of MAINLINE
C*
C*
C* Return to programs caller
C           DONE      BEGSR
C                     SETON                     LR
C                     RETRN
C                     ENDSR
C*
C* Calculate 4-byte aligned offset for next variable length record
C*
C           CALCVO    BEGSR
C           QUSBCD    ADD  12        BINARY
C                     DIV  4         BINARY
C                     MVR            BINARY
C           BINARY    IFEQ 0
C           QUSBCD    ADD  12        QUSBCB
C                     ELSE
C           4         SUB  BINARY    QUSBCB
C                     ADD  QUSBCD    QUSBCB
C                     ADD  12        QUSBCB
C                     END
C                     MOVEAQUSBC     REC,VO
C                     ADD  QUSBCB    VO
C                     ENDSR
O*
OQPRINT  E  106           ERREPT
O                                      'Attempt to register exit'
O                                      ' point failed: '
O                         QUSBND
OQPRINT  E  106           ERRPGM
O                                      'Attempt to add an exit'
O                                      ' program failed: '
O                         QUSBND
```

Example in ILE RPG: Registering exit points and adding exit programs

This ILE RPG program registers an exit point with the registration facility. After the successful completion of the registration, the program adds an exit program to the exit point.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*************************************************************
F*************************************************************
F*
F*  Program:      Register an Exit Point
F*                Add an Exit Program
```

```
F*
F*  Language:     ILE RPG
F*
F*  Description:  This program registers an exit point with the
F*                registration facility.  After the successful
F*                completion of the registration of the exit point,
F*                an exit program is added to the exit point.
F*
F*  APIs Used:    QusRegisterExitPoint - Register Exit Point
F*                QusAddExitProgram    - Add Exit Program
F*
F*****************************************************************
F*****************************************************************
F*
FQPRINT    O    F  132        PRINTER OFLIND(*INOF) USROPN
D*
D* Keyed Variable Length Record includes
D*
D/COPY QSYSINC/QRPGLESRC,QUS
D*
D* Error Code parameter include.  As this sample program
D* uses /COPY to include the error code structure, only the first
D* 16 bytes of the error code structure are available.  If the
D* application program needs to access the variable length
D* exception data for the error, the developer should physically
D* copy the QSYSINC include and modify the copied include to
D* define additional storage for the exception data.
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D*****************************************************************
D*Prototype for calling Register Exit Point API.
D*****************************************************************
D QUSREP05        C                   'QusRegisterExitPoint'
D*****************************************************************
D*Prototype for calling Add Exit Program API.
D*****************************************************************
D QUSAEPGM        C                   'QusAddExitProgram'
D*
D* Miscellaneous data
D*
DVARREC           DS
D NBR_RECS                        9B 0
D RECS                          1000
DV_OFFSET         S               9  0 INZ(1)
D*
DOVERLAYS         DS
D BINARY                          9B 0
D BINARY_C                        4    OVERLAY(BINARY)
D*
DEPNTNAME         S              20    INZ('EXAMPLE_EXIT_POINT')
DEPGM             S              20    INZ('EXAMPLEPGMEXAMPLELIB')
DEPGMDTA          S              25    INZ('EXAMPLE EXIT PROGRAM DATA')
DEPGMDTA_SZ       S               9B 0 INZ(%SIZE(EPGMDTA))
C*
C* Beginning of mainline
C*
C* Register the exit point with the registration facility.  If the
C* registration of the exit point is successful, add an exit
C* program to the exit point.
C*
C* Initialize the error code parameter.  To signal exceptions to
C* this program by the API, you need to set the bytes provided
C* field of the error code to zero.  Because this program has
C* exceptions sent back through the error code parameter, it sets
C* the bytes provided field to the number of bytes it gives the
C* API for the parameter.
```

```
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Set the exit point controls.  Each control field is passed to
C* the API using a variable length record.  Each record must
C* start on a 4-byte boundary.
C*
C* Set the total number of controls that are being specified on
C* the call.  This program lets the API take the default for the
C* controls that are not specified.
C*
C                   EVAL      NBR_RECS = 2
C*
C* Set the values for the two controls that are specified:
C*   Maximum number of exit programs = 10
C*   Exit point description = 'EXIT POINT EXAMPLE'
C*
C                   EVAL      QUSCK = 3
C                   EVAL      QUSLD = 4
C                   EVAL      BINARY = 10
C                   EVAL      %SUBST(RECS&#58;V_OFFSET+12) = BINARY_C
C                   EXSR      CALC_VOFF
C                   EVAL      QUSCK = 8
C                   EVAL      QUSLD = 50
C                   EVAL      %SUBST(RECS&#58;V_OFFSET+12:50) = 'EXIT +
C                              POINT EXAMPLE'
C                   EXSR      CALC_VOFF
C*
C* Call the API to register the exit point.
C*
C                   CALLB     QUSREP05
C                   PARM                    EPNTNAME
C                   PARM      'EXMP0100'    FORMAT            8
C                   PARM                    VARREC
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   OPEN      QPRINT
C                   EXCEPT    ERRAEPNT
C                   EXSR      DONE
C                   ENDIF
C*
C* If the call to register an exit point is successful, add
C* an exit program to the exit point.
C*
C* Set the total number of exit program attributes that are being
C* specified on the call.  This program lets the API take the
C* default for the attributes that are not specified.  Each
C* attribute record must be 4-byte aligned.
C*
C                   EVAL      NBR_RECS = 2
C                   EVAL      V_OFFSET = 1
C*
C* Set the values for the two attributes that are being specified:
C*   Replace exit program    = 1
C*   Exit program data CCSID = 37
C*
C                   EVAL      QUSCK = 4
C                   EVAL      QUSLD = 1
C                   EVAL      %SUBST(RECS&#58;V_OFFSET+12) = '1'
C                   EXSR      CALC_VOFF
C                   EVAL      QUSCK = 3
```

```
C                   EVAL      QUSLD = 4
C                   EVAL      BINARY = 37
C                   EVAL      %SUBST(RECS&#58;V_OFFSET+12) = BINARY_C
C                   EXSR      CALC_VOFF
C*
C* Call the API to add the exit program.
C*
C                   CALLB     QUSAEPGM
C                   PARM                    EPNTNAME
C                   PARM      'EXMP0100'    FORMAT
C                   PARM      1             BINARY
C                   PARM                    EPGM
C                   PARM                    EPGMDTA
C                   PARM                    EPGMDTA_SZ
C                   PARM                    VARREC
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   OPEN      QPRINT
C                   EXCEPT    ERRAEPGM
C                   EXSR      DONE
C                   ENDIF
C                   EXSR      DONE
C*
C* End of MAINLINE
C*
C*
C* Return to programs caller
C     DONE          BEGSR
C                   EVAL      *INLR = '1'
C                   RETURN
C                   ENDSR
C*
C* Calculate 4-byte aligned offset for next variable length record
C*
C     CALC_VOFF     BEGSR
C                   EVAL      BINARY = QUSLD + 12
C                   DIV       4             BINARY
C                   MVR                     BINARY
C                   IF        BINARY = 0
C                   EVAL      QUSLVR00 = (QUSLD + 12)
C                   ELSE
C                   EVAL      QUSLVR00 = (QUSLD + 12 + (4 - BINARY))
C                   END
C                   EVAL      %SUBST(RECS&#58;V_OFFSET:12) = QUSVR4
C                   EVAL      V_OFFSET = V_OFFSET + QUSLVR00
C                   ENDSR
O*
OQPRINT    E            ERRAEPNT       1  6
O                                              'Attempt to register exit -
O                                              point failed: '
O                       QUSEI
OQPRINT    E            ERRAEPGM       1  6
O                                              'Attempt to add exit -
O                                              program failed: '
O                       QUSEI
```

Example in ILE C: Removing exit programs and deregistering exit points

This ILE C program removes an exit program from an exit point. After the successful completion of the removal, the program deregisters the exit point from the registration facility.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*  PROGRAM:      Remove an Exit Program                             */
/*                Deregister an Exit Point                           */
/*                                                                   */
/*  LANGUAGE:     ILE C                                              */
/*                                                                   */
/*  DESCRIPTION:  This program removes an exit program and           */
/*                deregisters an exit point from the registration    */
/*                facility.                                          */
/*                                                                   */
/*  APIs USED:    QusRemoveExitProgram   - Remove Exit Program       */
/*                QusDeregisterExitPoint - Deregister Exit Point     */
/*                                                                   */
/*********************************************************************/
/*  NOTE:  This example uses APIs that are shipped with *EXCLUDE     */
/*         authority.  The user needs *USE authority to the service  */
/*         program QUSRGFA1 to use these APIs.                       */
/*********************************************************************/

/*********************************************************************/
/*                      Includes                                     */
/*********************************************************************/
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <qusrgfa1.h>
#include <qusec.h>
#include <qliept.h>

/*********************************************************************/
/*                      Structures                                   */
/*********************************************************************/

typedef struct {                      /* Error code                  */
 Qus_EC_t ec_fields;
 char     exception_data[100];
} error_code_struct;

/*********************************************************************/
/*                                                                   */
/*                              main                                 */
/*                                                                   */
/*********************************************************************/
int main()
{
  int  pgm_num=1;
  error_code_struct error_code;

  /*******************************************************************/
  /* Remove an exit program from the exit point and then deregister  */
  /* the exit point.  It is not necessary to remove exit programs    */
  /* from an exit point before deregistering the exit point.  It is  */
  /* done here only for illustration purposes.                       */
  /*******************************************************************/

  /*******************************************************************/
  /* Initialize the error code parameter.  To have exceptions        */
  /* signaled to this program by the API, set the bytes provided     */
  /* field of the code to zero.  This program has exceptions sent    */
  /* through the error code parameter; therefore, the bytes          */
  /* provided field is set to the number of bytes that this program  */
  /* gives the API for the parameter.                                */
  /*******************************************************************/
```

```
  error_code.ec_fields.Bytes_Provided=sizeof(error_code_struct);

  /*****************************************************************/
  /* Call the API to remove the exit program.                      */
  /*****************************************************************/
  QusRemoveExitProgram("EXAMPLE_EXIT_POINT  ",
                       "EXMP0100",
                       pgm_num,
                       &error_code);

  /*****************************************************************/
  /* If an exception occurs, the API returns the exception in the  */
  /* error code parameter.  The bytes available field is set to    */
  /* zero if no exception occurs and nonzero if an exception does  */
  /* occur.                                                        */
  /*****************************************************************/
  if (error_code.ec_fields.Bytes_Available != 0)
  {
     printf("ATTEMPT TO REMOVE EXIT PROGRAM FAILED WITH EXCEPTION: %.7s",
             error_code.ec_fields.Exception_Id);
     exit(1);
  }

  /*****************************************************************/
  /* If the call to remove the exit program is successful,         */
  /* deregister the exit point.                                    */
  /*****************************************************************/

  /*****************************************************************/
  /* Call the API to add the exit program.                         */
  /*****************************************************************/
  QusDeregisterExitPoint("EXAMPLE_EXIT_POINT  ",
                         "EXMP0100",
                         &error_code);

  /*****************************************************************/
  /* If an exception occurs, the API returns the exception in the  */
  /* error code parameter.  The bytes available field is set to    */
  /* zero if no exception occurs and nonzero if an exception does  */
  /* occur.                                                        */
  /*****************************************************************/
  if (error_code.ec_fields.Bytes_Available != 0)
  {
     printf("ATTEMPT TO DEREGISTER EXIT POINT FAILED WITH EXCEPTION: %.7s",
             error_code.ec_fields.Exception_Id);
     exit(1);
  }


} /* End program */
```

Example in OPM COBOL: Removing exit programs and deregistering exit points

This OPM COBOL program removes an exit program from an exit point. After the successful completion of the removal, the program deregisters the exit point from the registration facility.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
       IDENTIFICATION DIVISION.
      ****************************************************************
      ****************************************************************
      *
      *  Program:      Remove an Exit Program
      *                Deregister an Exit Point
      *
      *  Language:     OPM COBOL
```

```
*
*  Description:  This program removes an exit program and
*                deregisters an exit point from the registration
*                facility.
*
*  APIs Used:    QUSRMVEP - Remove Exit Program
*                QUSDRGPT - Deregister Exit Point
*
*****************************************************************
*
*****************************************************************
 PROGRAM-ID. REGFAC1.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION.
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Error message text
*
 01  BAD-EXIT-POINT.
     05  TEXT1        PIC X(41)
               VALUE "Attempt to deregister exit point failed: ".
     05  EXCEPTION-ID PIC X(07).
 01  BAD-EXIT-PGM.
     05  TEXT1        PIC X(39)
               VALUE "Attempt to remove exit program failed: ".
     05  EXCEPTION-ID PIC X(07).
*
* Miscellaneous elements
*
 01  MISC.
     05  PGM-NBR         PIC S9(09) VALUE 1 BINARY.
     05  EXIT-POINT-NAME PIC  X(20) VALUE "EXAMPLE_EXIT_POINT".
     05  FORMAT-NAME     PIC  X(08) VALUE "EXMP0100".
*
* Beginning of mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Remove an exit program from the exit point and then deregister
* the exit point.  It is not necessary to remove exit programs
* from an exit point before deregistering the exit point.  It is
* done here only for illustrative purposes.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
```

```
* API for the parameter.
*
     MOVE 16 TO BYTES-PROVIDED OF QUS-EC.
*
* Call the API to remove the exit program.
*
     CALL "QUSRMVEP" USING EXIT-POINT-NAME, FORMAT-NAME,
                           PGM-NBR, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           OPEN OUTPUT LISTING,
                           MOVE EXCEPTION-ID OF QUS-EC
                              TO EXCEPTION-ID OF BAD-EXIT-POINT,
                           WRITE LIST-LINE FROM BAD-EXIT-POINT,
                           STOP RUN.
*
* If the call to remove the exit program is successful,
* deregister the exit point.
*
* Call the API to deregister the exit point.
*
     CALL "QUSDRGPT" USING EXIT-POINT-NAME, FORMAT-NAME, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           OPEN OUTPUT LISTING,
                           MOVE EXCEPTION-ID OF QUS-EC
                              TO EXCEPTION-ID OF BAD-EXIT-PGM,
                           WRITE LIST-LINE FROM BAD-EXIT-PGM,
                           STOP RUN.
*
     STOP RUN.
*
* End of MAINLINE
*
```

Example in ILE COBOL: Removing exit programs and deregistering exit points

This ILE COBOL program removes an exit program from an exit point. After the successful completion of the removal, the program deregisters the exit point from the registration facility.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*  Program:      Remove an Exit Program
*                Deregister an Exit Point
*
*  Language:     ILE COBOL
*
*  Description:  This program removes an exit program and
*                deregisters an exit point from the registration
*                facility.
*
*  APIs Used:    QusRemoveExitProgram - Remove Exit Program
```

```
*               QusDeregisterExitPoint - Deregister Exit Point
*
****************************************************************
*
****************************************************************
 PROGRAM-ID. REGFAC3.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION.
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Error message text
*
 01  BAD-EXIT-POINT.
     05  TEXT1        PIC X(41)
               VALUE "Attempt to deregister exit point failed: ".
     05  EXCEPTION-ID PIC X(07).
 01  BAD-EXIT-PGM.
     05  TEXT1        PIC X(39)
               VALUE "Attempt to remove exit program failed: ".
     05  EXCEPTION-ID PIC X(07).
*
* Miscellaneous elements
*
 01  MISC.
     05  PGM-NBR         PIC S9(09) VALUE 1 BINARY.
     05  EXIT-POINT-NAME PIC  X(20) VALUE "EXAMPLE_EXIT_POINT".
     05  FORMAT-NAME     PIC  X(08) VALUE "EXMP0100".
*
* Beginning of mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Remove an exit program from the exit point and then deregister
* the exit point.  It is not necessary to remove exit programs
* from an exit point before deregistering the exit point.  It is
* done here only for illustrative purposes.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
* API for the parameter.
*
     MOVE 16 TO BYTES-PROVIDED OF QUS-EC.
*
* Call the API to remove the exit program.
*
```

```
     CALL PROCEDURE "QusRemoveExitProgram" USING
                           EXIT-POINT-NAME, FORMAT-NAME,
                           PGM-NBR, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           OPEN OUTPUT LISTING,
                           MOVE EXCEPTION-ID OF QUS-EC
                              TO EXCEPTION-ID OF BAD-EXIT-POINT,
                           WRITE LIST-LINE FROM BAD-EXIT-POINT,
                           STOP RUN.
*
* If the call to remove the exit program is successful,
* deregister the exit point.
*
* Call the API to deregister the exit point.
*
     CALL PROCEDURE "QusDeregisterExitPoint" USING
                          EXIT-POINT-NAME, FORMAT-NAME, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           OPEN OUTPUT LISTING,
                           MOVE EXCEPTION-ID OF QUS-EC
                              TO EXCEPTION-ID OF BAD-EXIT-PGM,
                           WRITE LIST-LINE FROM BAD-EXIT-PGM,
                           STOP RUN.
*
     STOP RUN.
*
* End of MAINLINE
*
```

Example in OPM RPG: Removing exit programs and deregistering exit points

This OPM RPG program removes an exit program from an exit point. After the successful completion of the removal, the program deregisters the exit point from the registration facility.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*****************************************************************
F*****************************************************************
F*
F*  Program:      Remove an Exit Program
F*                Deregister an Exit Point
F*
F*  Language:     OPM RPG
F*
F*  Description:  This program removes an exit program and
F*                deregisters an exit point from the registration
F*                facility.
F*
F*  APIs Used:    QUSRMVEP - Remove Exit Program
F*                QUSDRGPT - Deregister Exit Point
F*
F*****************************************************************
F*****************************************************************
F*
```

```
FQPRINT  O   F     132            PRINTER                        UC
I*
I* Error Code parameter include.  As this sample program
I* uses /COPY to include the error code structure, only the first
I* 16 bytes of the error code structure are available.  If the
I* application program needs to access the variable length
I* exception data for the error, the developer should physically
I* copy the QSYSINC include and modify the copied include to
I* define additional storage for the exception data.
I*
I/COPY QSYSINC/QRPGSRC,QUSEC
I*
I*
I* Miscellaneous data
I*
I            DS
I                                     B   1   40PGMNBR
I I            'EXAMPLE_EXIT_POINT  '    5  24 EPNTNM
C*
C* Beginning of mainline
C*
C* Remove an exit program from the exit point and then deregister
C* the exit point.  It is not necessary to remove exit programs
C* from an exit point before deregistering the exit point.  It is
C* done here only for illustrative purposes.
C*
C* Initialize the error code parameter.  To signal exceptions to
C* this program by the API, you need to set the bytes provided
C* field of the error code to zero.  Because this program has
C* exceptions sent back through the error code parameter, it sets
C* the bytes provided field to the number of bytes it gives the
C* API for the parameter.
C*
C                     Z-ADD16        QUSBNB
C*
C* Call the API to remove the exit program.
C*
C                     CALL 'QUSRMVEP'
C                     PARM           EPNTNM
C                     PARM 'EXMP0100'FORMAT  8
C                     PARM 1         PGMNBR
C                     PARM           QUSBN
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C           QUSBNC    IFGT 0
C                     OPEN QPRINT
C                     EXCPTERRPGM
C                     EXSR DONE
C                     ENDIF
C*
C* If the call to remove the exit program is successful,
C* deregister the exit point.
C*
C* Call the API to deregister the exit point.
C*
C                     CALL 'QUSDRGPT'
C                     PARM           EPNTNM
C                     PARM 'EXMP0100'FORMAT
C                     PARM           QUSBN
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
```

```
C* exception does occur.
C*
C           QUSBNC    IFGT 0
C                     OPEN QPRINT
C                     EXCPTERREPT
C                     EXSR DONE
C                     ENDIF
C                     EXSR DONE
C*
C* End of MAINLINE
C*
C*
C* Return to programs caller
C           DONE      BEGSR
C                     SETON                     LR
C                     RETRN
C                     ENDSR
O*
OQPRINT  E  106           ERREPT
O                                     'Attempt to deregister '
O                                     'exit point failed: '
O                         QUSBND
OQPRINT  E  106           ERRPGM
O                                     'Attempt to remove exit '
O                                     'program failed: '
O                         QUSBND
```

Example in ILE RPG: Removing exit programs and deregistering exit points

This ILE RPG program removes an exit program from an exit point. After the successful completion of the removal, the program deregisters the exit point from the registration facility.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*****************************************************************
F*****************************************************************
F*
F*  Program:      Remove an Exit Program
F*                Deregister an Exit Point
F*
F*  Language:     ILE RPG
F*
F*  Description:  This program removes an exit program and
F*                deregisters an exit point from the registration
F*                facility.
F*
F*  APIs Used:    QusRemoveExitProgram   - Remove Exit Program
F*                QusDeregisterExitPoint - Deregister Exit Point
F*
F*****************************************************************
F*****************************************************************
F*
FQPRINT    O    F  132        PRINTER OFLIND(*INOF) USROPN
D*
D* Error Code parameter include.  As this sample program
D* uses /COPY to include the error code structure, only the first
D* 16 bytes of the error code structure are available.  If the
D* application program needs to access the variable length
D* exception data for the error, the developer should physically
D* copy the QSYSINC include and modify the copied include to
D* define additional storage for the exception data.
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D*****************************************************************
D*Prototype for calling Deregister Exit Point API.
```

```
D*****************************************************************
D QUSDEP          C                   'QusDeregisterExitPoint'
D*****************************************************************
D*Prototype for calling Remove Exit Program API.
D*****************************************************************
D QUSREPGM        C                   'QusRemoveExitProgram'
D*
D* Miscellaneous data
D*
DPGM_NBR                         9B 0
DEPNTNAME         S             20    INZ('EXAMPLE_EXIT_POINT')
C*
C* Beginning of mainline
C*
C* Remove an exit program from the exit point and then deregister
C* the exit point.  It is not necessary to remove exit programs
C* from an exit point before deregistering the exit point.  It is
C* done here only for illustrative purposes.
C*
C* Initialize the error code parameter.  To signal exceptions to
C* this program by the API, you need to set the bytes provided
C* field of the error code to zero.  Because this program has
C* exceptions sent back through the error code parameter, it sets
C* the bytes provided field to the number of bytes it gives the
C* API for the parameter.
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Call the API to remove the exit program.
C*
C                   CALLB     QUSREPGM
C                   PARM                    EPNTNAME
C                   PARM      'EXMP0100'    FORMAT            8
C                   PARM      1             PGM_NBR
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   OPEN      QPRINT
C                   EXCEPT    ERRAEPGM
C                   EXSR      DONE
C                   ENDIF
C*
C* If the call to remove the exit program is successful,
C* deregister the exit point.
C*
C* Call the API to deregister the exit point.
C*
C                   CALLB     QUSDEP
C                   PARM                    EPNTNAME
C                   PARM      'EXMP0100'    FORMAT
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   OPEN      QPRINT
C                   EXCEPT    ERRAEPNT
C                   EXSR      DONE
C                   ENDIF
```

```
C                   EXSR      DONE
C*
C* End of MAINLINE
C*
C*
C* Return to programs caller
C     DONE          BEGSR
C                   EVAL      *INLR = '1'
C                   RETURN
C                   ENDSR
O*
OQPRINT    E            ERRAEPNT       1  6
O                                              'Attempt to deregister -
O                                              exit point failed: '
O                       QUSEI
OQPRINT    E            ERRAEPGM       1  6
O                                              'Attempt to remove exit -
O                                              program failed: '
O                       QUSEI
```

Example in ILE C: Retrieving exit point and exit program information

This ILE C program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

The Retrieve Exit Information (QusRetrieveExitInformation) API returns a continuation handle when it has more information to return than what can fit in the receiver variable.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*  PROGRAM:      Retrieve Exit Point and Exit Program Information  */
/*                                                                   */
/*  LANGUAGE:     ILE C                                              */
/*                                                                   */
/*  DESCRIPTION:  This program retrieves exit point and exit         */
/*                program information.  After retrieving the         */
/*                exit point information, the program resolves to    */
/*                each associated exit program and calls each exit   */
/*                program.                                           */
/*                                                                   */
/*  APIs USED:   QusRetrieveExitInformation - Retrieve Exit          */
/*                                            Information            */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                     Includes                                      */
/*********************************************************************/
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <except.h>
#include <qusrgfa2.h>
#include <qusec.h>
#include <qmhchgem.h>
#include <miptrnam.h>
#include <qliept.h>

/*********************************************************************/
/*                     Prototypes                                    */
/*********************************************************************/
typedef void Pgm_OS(void *arg,...);
#pragma linkage(Pgm_OS,OS)
```

```
/*********************************************************************/
/*                     Structures                                    */
/*********************************************************************/

typedef struct {                      /* Error code                  */
 Qus_EC_t ec_fields;
 char     exception_data[100];
} error_code_struct;

/*********************************************************************/
/*   FUNCTION NAME:  RSLVSP_PGM_HDLR                                 */
/*                                                                   */
/*   FUNCTION :      This function handles all exceptions that       */
/*                   may occur while resolving to the exit           */
/*                   program.                                        */
/*                                                                   */
/*   INPUT:          Interrupt handler information                   */
/*                                                                   */
/*   OUTPUT:         NONE                                            */
/*                                                                   */
/*********************************************************************/
void RSLVSP_PGM_HDLR(_INTRPT_Hndlr_Parms_T *errmsg)
{
  error_code_struct Error_Code;

  /*******************************************************************/
  /* Set the rsl_ok indicator to not valid.                          */
  /*******************************************************************/
  int *rsl_ok = (int *)(errmsg>Com_Area);
  *rsl_ok = 0;

  /*******************************************************************/
  /* Let message handler know that the program handled the message   */
  /* and to remove it from the job log.                              */
  /*******************************************************************/
  Error_Code.ec_fields.Bytes_Provided=0;
  QMHCHGEM(&(errmsg>Target),
           0,
           (char *)&errmsg>Msg_Ref_Key,
           "*REMOVE   ",
           "",
           0,
           &Error_Code);

}

/*********************************************************************/
/*   FUNCTION NAME:  Call_Exit_Program                               */
/*                                                                   */
/*   FUNCTION :      This function calls the exit programs that      */
/*                   were retrieved from the registration facility   */
/*                   repository.                                     */
/*                                                                   */
/*   INPUT:          Information retrieved                           */
/*                                                                   */
/*   OUTPUT:         NONE                                            */
/*                                                                   */
/*********************************************************************/
void Call_Exit_Program(char *rcv_var)
{
  int  num_exit_pgms,
       i;
  char exit_pgm_name[10],
       exit_pgm_lib[10],
       info_for_exit_pgm[10],
       *rcv_ptr;
  volatile int rsl_ok;
```

```
  Pgm_OS *exit_pgm_ptr;

  /*****************************************************************/
  /* Save the number of exit programs returned and set the pointer  */
  /* to point to the first exit program entry.                      */
  /*****************************************************************/
  rcv_ptr=rcv_var;
  num_exit_pgms=((Qus_EXTI0200_t *)rcv_ptr)>Number_Programs_Returned;
  rcv_ptr += ((Qus_EXTI0200_t *)rcv_ptr)>Offset_Program_Entry;
  rsl_ok=1;

  for (i=0; i<num_exit_pgms; i++)
  {
    memcpy(exit_pgm_name,
             ((Qus_EXTI0200_Entry_t *)rcv_ptr)>Program_Name,10);
    memcpy(exit_pgm_lib,
             ((Qus_EXTI0200_Entry_t *)rcv_ptr)>Program_Library,10);

    /*****************************************************************/
    /* Resolve to the exit program. If an error occurs on the       */
    /* resolve operation to the library, the rsl_ok indicator is    */
    /* set to failed in the RSL_PGM_HDLR exception handler.         */
    /* The rslvsp MI instruction signals all errors to this         */
    /* program; therefore, enable the exception handler to capture  */
    /* any errors that may occur.                                   */
    /*****************************************************************/
    #pragma exception_handler (RSLVSP_PGM_HDLR,rsl_ok,0,_C2_MH_ESCAPE)

    exit_pgm_ptr=((Pgm_OS *)rslvsp(_Program,
                                   exit_pgm_name,
                                   exit_pgm_lib,
                                   _AUTH_POINTER));
    #pragma disable_handler

    /*****************************************************************/
    /* If the resolve operation is successful, call the exit        */
    /* program.  If not, move on to the next exit program.          */
    /*****************************************************************/
    if (rsl_ok)
    {
       exit_pgm_ptr(info_for_exit_pgm);
    }

    /*****************************************************************/
    /* Set the receiver variable to point to the next exit program  */
    /* that is returned.                                            */
    /*****************************************************************/
    rsl_ok=1;
    rcv_ptr=rcv_var +
                 ((Qus_EXTI0200_Entry_t *)rcv_ptr)>Offset_Next_Entry;
  }
}

/*********************************************************************/
/*                                                                   */
/*                              main                                 */
/*                                                                   */
/*********************************************************************/
void main()
{
  int  sel_criteria=0,
       len_rcv_variable=3500,
       exit_pgm_num=-1;
  char continuation_hdl[16],
       rcv_variable[3500],
       *rcv_ptr;
  error_code_struct error_code;
```

```
/*******************************************************************/
/* Retrieve the exit point information first.  If the current      */
/* number of exit programs is not zero, retrieve the exit          */
/* programs.  It is not necessary to call for the exit point       */
/* information to determine if the exit point has any exit         */
/* programs. It is done here for illustration purposes only.       */
/* You can make one call to the API for the exit program           */
/* information and check the number of exit program entries        */
/* returned field to see if there are any exit programs to call.  */
/*******************************************************************/

/*******************************************************************/
/* Initialize the error code to inform the API that all            */
/* exceptions should be returned through the error code parameter.*/
/*******************************************************************/
error_code.ec_fields.Bytes_Provided=sizeof(error_code_struct);

/*******************************************************************/
/* Blank out the continuation handle to let the API know that this*/
/* is a first attempt at the retrieve operation.                   */
/*******************************************************************/
memset(continuation_hdl,' ',16);

/*******************************************************************/
/* Call the API to retrieve the exit point information.            */
/*******************************************************************/
QusRetrieveExitInformation(continuation_hdl,
                           &rcv_variable,
                           len_rcv_variable,
                           "EXTI0100",
                           "EXAMPLE_EXIT_POINT  ",
                           "EXMP0100",
                           exit_pgm_num,
                           &sel_criteria,
                           &error_code);

/*******************************************************************/
/* If an exception occurs, the API returns the exception in the    */
/* error code parameter.  The bytes available field is set to      */
/* zero if no exception occurs and nonzero if an exception does    */
/* occur.                                                          */
/*******************************************************************/
if (error_code.ec_fields.Bytes_Available != 0)
{
   printf("ATTEMPT TO RETRIEVE INFORMATION FAILED WITH EXCEPTION: %.7s",
           error_code.ec_fields.Exception_Id);
   exit(1);
}

/*******************************************************************/
/* If the call to retrieve exit point information is successful,  */
/* check to see if there are any exit programs to call.            */
/*******************************************************************/
rcv_ptr=rcv_variable;
rcv_ptr += ((Qus_EXTI0100_t *)rcv_ptr)->Offset_Exit_Point_Entry;

if (((Qus_EXTI0100_Entry_t *)rcv_ptr)->Number_Exit_Programs != 0)
{

  /*****************************************************************/
  /* Blank out the continuation handle to let the API know that    */
  /* this is a first attempt at the retrieve operation.            */
  /*****************************************************************/
  memset(continuation_hdl,' ',16);

  /*****************************************************************/
```

```
/* Call the API to retrieve the exit program information.        */
/*****************************************************************/
QusRetrieveExitInformation(continuation_hdl,
                           &rcv_variable,
                           len_rcv_variable,
                           "EXTI0200",
                           "EXAMPLE_EXIT_POINT  ",
                           "EXMP0100",
                           exit_pgm_num,
                           &sel_criteria,
                           &error_code);

/*****************************************************************/
/* Verify that the call to the API is successful.                */
/*****************************************************************/
if (error_code.ec_fields.Bytes_Available != 0)
{
   printf("ATTEMPT TO RETRIEVE EXIT PROGRAMS FAILED WITH EXCEPTION:\
           %.7s", error_code.ec_fields.Exception_Id);
   exit(1);
}

/*****************************************************************/
/* If the call is successful, call the exit programs.            */
/*****************************************************************/
Call_Exit_Program(rcv_variable);

/*****************************************************************/
/* If the continuation handle field in the receiver variable is  */
/* not set to blanks, the API has more information to return     */
/* than what could fit in the receiver variable.                 */
/*****************************************************************/
rcv_ptr=rcv_variable;

while (memcmp(((Qus_EXTI0200_t *)rcv_ptr)->Continue_Handle,
       "                ",16)!=0)
{
  memcpy(continuation_hdl,
         ((Qus_EXTI0200_t *)rcv_ptr)>Continue_Handle,16);

  /***************************************************************/
  /* Call the API to retrieve the exit program information.      */
  /***************************************************************/
  QusRetrieveExitInformation(continuation_hdl,
                             &rcv_variable,
                             len_rcv_variable,
                             "EXTI0200",
                             "EXAMPLE_EXIT_POINT  ",
                             "EXMP0100",
                             exit_pgm_num,
                             &sel_criteria,
                             &error_code);
  /***************************************************************/
  /* Verify that the call to the API is successful.              */
  /***************************************************************/
  if (error_code.ec_fields.Bytes_Available != 0)
  {
     printf("RETRIEVE EXIT PROGRAMS FAILED WITH EXCEPTION: %.7s",
             error_code.ec_fields.Exception_Id);
     exit(1);
  }

  /***************************************************************/
  /* If the call is successful, call the exit programs.          */
  /* The receiver variable offers enough room for a minimum of   */
  /* one exit program entry because the receiver variable was    */
  /* declared as 3500 bytes.  Therefore, this example only       */
```

```
      /* checks the number of exit programs returned field.  If the  */
      /* receiver variable were not large enough to hold at least    */
      /* one entry, the bytes available field would need to be       */
      /* checked as well as the number of exit programs returned     */
      /* field.  If the number of exit programs returned field is    */
      /* set to zero and the bytes available field is greater than   */
      /* the bytes returned field, the API had at least one exit     */
      /* program entry to return but was unable to because the       */
      /* receiver variable was too small.                            */
      /*************************************************************/
      Call_Exit_Program(rcv_variable);
    } /* While continuation handle not set to blanks */
  } /* Number of exit programs not equal to zero */

} /* End program */
```

Related tasks:

"Continuation handle" on page 76
When a call to an API is made and the API has more information to return than what can fit in the receiver variable or the user space, the API returns a continuation handle, which is used to mark the last value put in the receiver variable or the user space.

Related reference:

"Example in OPM COBOL: Retrieving exit point and exit program information"
This OPM COBOL program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

"Example in ILE COBOL: Retrieving exit point and exit program information" on page 221
This ILE COBOL program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

"Example in OPM RPG: Retrieving exit point and exit program information" on page 225
This OPM RPG program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

"Example in ILE RPG: Retrieving exit point and exit program information" on page 229
This ILE RPG program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

Example in OPM COBOL: Retrieving exit point and exit program information

This OPM COBOL program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

The Retrieve Exit Information (QUSRTVEI) API returns a continuation handle when it has more information to return than what can fit in the receiver variable.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
       IDENTIFICATION DIVISION.
      *************************************************************
      *************************************************************
      *
      *  Program:      Retrieve Exit Point and Exit Program Information
      *
      *  Language:     OPM COBOL
      *
      *  Description:  This program retrieves exit point and exit
      *                program information.  After retrieving the
      *                exit point information, the program calls each
      *                exit program.
      *
      *  APIs Used:    QUSCRTUS - Create User Space
      *                QUSPTRUS - Retrieve Pointer to User Space
      *                QUSRTVEI - Retrieve Exit Information
```

```
*
*****************************************************************
*****************************************************************
*
 PROGRAM-ID. REGFAC2.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION.
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Error message text
*
 01  BAD-EXIT-POINT.
     05  TEXT1        PIC X(40)
               VALUE "Attempt to retrieve information failed: ".
     05  EXCEPTION-ID PIC X(07).
 01  BAD-EXIT-PGM.
     05  TEXT1        PIC X(42)
              VALUE "Attempt to retrieve Exit Programs failed: ".
     05  EXCEPTION-ID PIC X(07).
 01  BAD-CREATE.
     05  TEXT1        PIC X(37)
              VALUE "Allocation of RCVVAR storage failed: ".
     05  EXCEPTION-ID PIC X(07).
*
* Miscellaneous elements
*
 01  MISC.
     05  EXIT-POINT-NAME PIC  X(20) VALUE "EXAMPLE_EXIT_POINT".
     05  EXIT-PGM-NBR    PIC S9(09) VALUE -1 BINARY.
     05  EXIT-PARAMETERS PIC  X(10).
     05  FORMAT-NAME     PIC  X(08) VALUE "EXTI0100".
     05  FORMAT-NAME-1   PIC  X(08) VALUE "EXTI0200".
     05  FORMAT-NAME-2   PIC  X(08) VALUE "EXMP0100".
     05  NBR-OF-SELECT-CRITERIA PIC S9(09) VALUE 0 BINARY.
     05  CONTINUATION-HDL PIC X(16).
     05  BASE-POINTER    POINTER.
     05  INFO-POINTER    POINTER.
     05  SPACE-NAME      PIC  X(20) VALUE "RCVVAR    QTEMP     ".
     05  SPACE-ATTR      PIC  X(10).
     05  SPACE-SIZE      PIC S9(09) VALUE 3500 BINARY.
     05  SPACE-VALUE     PIC  X(01) VALUE X"00".
     05  SPACE-AUTH      PIC  X(10) VALUE "*USE".
     05  SPACE-TEXT      PIC  X(50).
     05  SPACE-REPLACE   PIC  X(10) VALUE "*NO".
     05  SPACE-DOMAIN    PIC  X(10) VALUE "*USER".
*
 LINKAGE SECTION.
*
* Variable to hold results of QUSRTVEI.  The storage for this
```

```
* variable will be allocated by way of a User Space.
*
 01  RCVVAR          PIC  X(3500).
*
* Registration Facility API include.  These includes will be
* mapped over the RCVVAR (User Space) previously defined.
*
 COPY QUSREG OF QSYSINC-QLBLSRC.
*
* Beginning of mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Retrieve the exit point information first.  If the current
* number of exit programs is not zero, retrieve the exit
* programs.  It is not necessary to call for the exit point
* information to determine if the exit point has any exit
* programs.  It is done here for illustrative purposes only.
* You can make one call to the API for the exit program
* information and check the number of exit program entries
* returned field to see if there are any exit programs to call.
*
* Initialize the error code to inform the API that all
* exceptions should be returned through the error code parameter.
*
     MOVE 16 TO BYTES-PROVIDED OF QUS-EC.
*
* Create a User Space for RCVVAR.
*
     CALL "QUSCRTUS" USING SPACE-NAME, SPACE-ATTR, SPACE-SIZE,
                           SPACE-VALUE, SPACE-AUTH, SPACE-TEXT,
                           SPACE-REPLACE, QUS-EC, SPACE-DOMAIN.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
        IF EXCEPTION-ID OF QUS-EC = "CPF9870"
           CONTINUE
        ELSE
            OPEN OUTPUT LISTING,
            MOVE EXCEPTION-ID OF QUS-EC
            TO EXCEPTION-ID OF BAD-CREATE,
            WRITE LIST-LINE FROM BAD-CREATE,
            STOP RUN.
*
* Assign BASE-POINTER to address RCVVAR
*
     CALL "QUSPTRUS" USING SPACE-NAME, BASE-POINTER, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           OPEN OUTPUT LISTING,
                           MOVE EXCEPTION-ID OF QUS-EC
                             TO EXCEPTION-ID OF BAD-CREATE,
                           WRITE LIST-LINE FROM BAD-CREATE,
                           STOP RUN.
*
     SET ADDRESS OF RCVVAR TO BASE-POINTER.
*
```

```
* Blank out the continuation handle to let the API know that this
* is a first attempt at the retrieve operation.
*
     MOVE SPACES TO CONTINUATION-HDL.
*
* Call the API to retrieve the exit programs
*
     CALL "QUSRTVEI" USING CONTINUATION-HDL, RCVVAR,
                           BY CONTENT LENGTH OF RCVVAR,
                           FORMAT-NAME OF MISC,
                           EXIT-POINT-NAME OF MISC,
                           FORMAT-NAME-2, EXIT-PGM-NBR,
                           NBR-OF-SELECT-CRITERIA, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           OPEN OUTPUT LISTING,
                           MOVE EXCEPTION-ID OF QUS-EC
                             TO EXCEPTION-ID OF BAD-EXIT-POINT,
                           WRITE LIST-LINE FROM BAD-EXIT-POINT,
                           STOP RUN.
*
* If the call to retrieve exit point information is successful,
* check to see if there are any exit programs to call.
*
     SET ADDRESS OF QUS-EXTI0100 TO BASE-POINTER.
     SET ADDRESS OF QUS-EXTI0200 TO BASE-POINTER.
*
     IF NUMBER-POINTS-RETURNED OF QUS-EXTI0100 > 0
        SET ADDRESS OF QUS-EXTI0100-ENTRY TO
            ADDRESS OF RCVVAR((OFFSET-EXIT-POINT-ENTRY OF
                               QUS-EXTI0100 + 1):)
     ELSE STOP RUN.
*
     IF NUMBER-EXIT-PROGRAMS OF QUS-EXTI0100-ENTRY > 0
*
* There are some exit programs to call.  Blank out the continuation
* handle to let the API know that this is a first attempt at the
* retrieve operation.
*
        MOVE SPACES TO CONTINUATION-HDL,
*
* Call the exit programs
*
        PERFORM CALL-EXIT-PROGRAMS,
*
* If the continuation handle field in the receiver variable is
* not set to blanks, the API has more information to return than
* what could fit in the receiver variable.  Call the API for
* more exit programs to call.
*
        PERFORM UNTIL CONTINUE-HANDLE OF QUS-EXTI0200 = SPACES
          MOVE CONTINUE-HANDLE OF QUS-EXTI0200
                                 TO CONTINUATION-HDL,
          PERFORM CALL-EXIT-PROGRAMS,
          END-PERFORM.
*
     STOP RUN.
*
* End of MAINLINE
*
*
* Process exit programs in receiver variable
```

```
*
 CALL-EXIT-PROGRAMS.
*
* Call the API to retrieve the exit program information
*
     CALL "QUSRTVEI" USING CONTINUATION-HDL, RCVVAR,
                           BY CONTENT LENGTH OF RCVVAR,
                           FORMAT-NAME-1,
                           EXIT-POINT-NAME OF MISC,
                           FORMAT-NAME-2, EXIT-PGM-NBR,
                           NBR-OF-SELECT-CRITERIA, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           OPEN OUTPUT LISTING,
                           MOVE EXCEPTION-ID OF QUS-EC
                             TO EXCEPTION-ID OF BAD-EXIT-PGM,
                           WRITE LIST-LINE FROM BAD-EXIT-PGM,
                           STOP RUN.
*
* If the call to retrieve exit program information is successful,
* check to see if there are any exit programs to call.
*
* The receiver variable offers enough room for a minimum of one
* exit program entry because the receiver variable was declared
* as 3500 bytes.  Therefore, this example only checks the
* number of exit programs returned field.  If the receiver
* variable were not large enough to hold at least one entry,
* the bytes available field would need to be checked as well as
* the number of exit programs returned field.  If the number of
* exit programs returned field is set to zero and the bytes
* available field is greater than the bytes returned field, the
* API had at least one exit program entry to return but was
* unable to because the receiver variable was too small.
*
     SET ADDRESS OF QUS-EXTI0200-ENTRY
         TO ADDRESS OF RCVVAR(OFFSET-PROGRAM-ENTRY
                                 OF QUS-EXTI0200 + 1:).
     PERFORM CALL-PGMS
             NUMBER-PROGRAMS-RETURNED OF QUS-EXTI0200 TIMES.
*
 CALL-PGMS.
*
* Call the exit program while ignoring failures on the call
*
     CALL PROGRAM-NAME OF QUS-EXTI0200-ENTRY USING
                                EXIT-PARAMETERS
                                ON EXCEPTION CONTINUE.
*
* Address the next exit program entry
*
     SET ADDRESS OF QUS-EXTI0200-ENTRY
             TO ADDRESS OF RCVVAR(OFFSET-NEXT-ENTRY
                                  OF QUS-EXTI0200-ENTRY + 1:).
```

Related tasks:

"Continuation handle" on page 76
When a call to an API is made and the API has more information to return than what can fit in the receiver variable or the user space, the API returns a continuation handle, which is used to mark the last value put in the receiver variable or the user space.

Related reference:

“Example in ILE C: Retrieving exit point and exit program information” on page 211
This ILE C program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

Example in ILE COBOL: Retrieving exit point and exit program information

This ILE COBOL program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

The Retrieve Exit Information (QusRetrieveExitInformation) API returns a continuation handle when it has more information to return than what can fit in the receiver variable.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 571.

```
 IDENTIFICATION DIVISION.
*****************************************************************
*****************************************************************
*
*  Program:      Retrieve Exit Point and Exit Program Information
*
*  Language:     ILE COBOL
*
*  Description:  This program retrieves exit point and exit
*                program information.  After retrieving the
*                exit point information, the program calls each
*                exit program.
*
*  APIs Used:    QUSCRTUS - Create User Space
*                QUSPTRUS - Retrieve Pointer to User Space
*                QusRetrieveExitInformation - Retrieve Exit
*                                             Information
*
*****************************************************************
*****************************************************************
*
 PROGRAM-ID. REGFAC2.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION.
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Error message text
*
 01  BAD-EXIT-POINT.
     05  TEXT1        PIC X(40)
                 VALUE "Attempt to retrieve information failed: ".
     05  EXCEPTION-ID PIC X(07).
 01  BAD-EXIT-PGM.
```

```
     05  TEXT1        PIC X(42)
              VALUE "Attempt to retrieve Exit Programs failed: ".
     05  EXCEPTION-ID PIC X(07).
 01  BAD-CREATE.
     05  TEXT1        PIC X(37)
              VALUE "Allocation of RCVVAR storage failed: ".
     05  EXCEPTION-ID PIC X(07).
*
* Miscellaneous elements
*
 01  MISC.
     05  EXIT-POINT-NAME PIC  X(20) VALUE "EXAMPLE_EXIT_POINT".
     05  EXIT-PGM-NBR    PIC S9(09) VALUE -1 BINARY.
     05  EXIT-PARAMETERS PIC  X(10).
     05  FORMAT-NAME     PIC  X(08) VALUE "EXTI0100".
     05  FORMAT-NAME-1   PIC  X(08) VALUE "EXTI0200".
     05  FORMAT-NAME-2   PIC  X(08) VALUE "EXMP0100".
     05  NBR-OF-SELECT-CRITERIA PIC S9(09) VALUE 0 BINARY.
     05  CONTINUATION-HDL PIC X(16).
     05  BASE-POINTER    POINTER.
     05  INFO-POINTER    POINTER.
     05  SPACE-NAME      PIC  X(20) VALUE "RCVVAR    QTEMP     ".
     05  SPACE-ATTR      PIC  X(10).
     05  SPACE-SIZE      PIC S9(09) VALUE 3500 BINARY.
     05  SPACE-VALUE     PIC  X(01) VALUE X"00".
     05  SPACE-AUTH      PIC  X(10) VALUE "*USE".
     05  SPACE-TEXT      PIC  X(50).
     05  SPACE-REPLACE   PIC  X(10) VALUE "*NO".
     05  SPACE-DOMAIN    PIC  X(10) VALUE "*USER".
*
 LINKAGE SECTION.
*
* Variable to hold results of QusRetrieveExitInformation.  The
* storage for this variable will be allocated by way of a User
* Space.
*
 01  RCVVAR          PIC  X(3500).
*
* Registration Facility API include.  These includes will be
* mapped over the RCVVAR (User Space) previously defined.
*
 COPY QUSREG OF QSYSINC-QLBLSRC.
*
* Beginning of mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Retrieve the exit point information first.  If the current
* number of exit programs is not zero, retrieve the exit
* programs.  It is not necessary to call for the exit point
* information to determine if the exit point has any exit
* programs.  It is done here for illustrative purposes only.
* You can make one call to the API for the exit program
* information and check the number of exit program entries
* returned field to see if there are any exit programs to call.
*
* Initialize the error code to inform the API that all
* exceptions should be returned through the error code parameter.
*
     MOVE 16 TO BYTES-PROVIDED OF QUS-EC.
*
* Create a User Space for RCVVAR.
*
     CALL "QUSCRTUS" USING SPACE-NAME, SPACE-ATTR, SPACE-SIZE,
                           SPACE-VALUE, SPACE-AUTH, SPACE-TEXT,
                           SPACE-REPLACE, QUS-EC, SPACE-DOMAIN.
```

```
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
        IF EXCEPTION-ID OF QUS-EC = "CPF9870"
           CONTINUE
        ELSE
            OPEN OUTPUT LISTING,
            MOVE EXCEPTION-ID OF QUS-EC
            TO EXCEPTION-ID OF BAD-CREATE,
            WRITE LIST-LINE FROM BAD-CREATE,
            STOP RUN.
*
* Assign BASE-POINTER to address RCVVAR
*
     CALL "QUSPTRUS" USING SPACE-NAME, BASE-POINTER, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                            OPEN OUTPUT LISTING,
                            MOVE EXCEPTION-ID OF QUS-EC
                              TO EXCEPTION-ID OF BAD-CREATE,
                            WRITE LIST-LINE FROM BAD-CREATE,
                            STOP RUN.
*
     SET ADDRESS OF RCVVAR TO BASE-POINTER.
*
* Blank out the continuation handle to let the API know that this
* is a first attempt at the retrieve operation.
*
     MOVE SPACES TO CONTINUATION-HDL.
*
* Call the API to retrieve the exit programs
*
     CALL PROCEDURE "QusRetrieveExitInformation" USING
                            CONTINUATION-HDL,
                            RCVVAR,
                            BY CONTENT LENGTH OF RCVVAR,
                            FORMAT-NAME OF MISC,
                            EXIT-POINT-NAME OF MISC,
                            FORMAT-NAME-2, EXIT-PGM-NBR,
                            NBR-OF-SELECT-CRITERIA, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                            OPEN OUTPUT LISTING,
                            MOVE EXCEPTION-ID OF QUS-EC
                              TO EXCEPTION-ID OF BAD-EXIT-POINT,
                            WRITE LIST-LINE FROM BAD-EXIT-POINT,
                            STOP RUN.
*
* If the call to retrieve exit point information is successful,
* check to see if there are any exit programs to call.
*
     SET ADDRESS OF QUS-EXTI0100 TO BASE-POINTER.
     SET ADDRESS OF QUS-EXTI0200 TO BASE-POINTER.
```

```
*
     IF NUMBER-POINTS-RETURNED OF QUS-EXTI0100 > 0
        SET ADDRESS OF QUS-EXTI0100-ENTRY TO
            ADDRESS OF RCVVAR((OFFSET-EXIT-POINT-ENTRY OF
                               QUS-EXTI0100 + 1):)
     ELSE STOP RUN.
*
     IF NUMBER-EXIT-PROGRAMS OF QUS-EXTI0100-ENTRY > 0
*
* There are some exit programs to call.  Blank out the continuation
* handle to let the API know that this is a first attempt at the
* retrieve operation.
*
        MOVE SPACES TO CONTINUATION-HDL,
*
* Call the exit programs
*
        PERFORM CALL-EXIT-PROGRAMS,
*
* If the continuation handle field in the receiver variable is
* not set to blanks, the API has more information to return than
* what could fit in the receiver variable.  Call the API for
* more exit programs to call.
*
        PERFORM UNTIL CONTINUE-HANDLE OF QUS-EXTI0200 = SPACES
          MOVE CONTINUE-HANDLE OF QUS-EXTI0200
                                  TO CONTINUATION-HDL,
          PERFORM CALL-EXIT-PROGRAMS,
          END-PERFORM.
*
     STOP RUN.
*
* End of MAINLINE
*
*
* Process exit programs in receiver variable
*
 CALL-EXIT-PROGRAMS.
*
* Call the API to retrieve the exit program information
*
     CALL PROCEDURE "QusRetrieveExitInformation" USING
                           CONTINUATION-HDL, RCVVAR,
                           BY CONTENT LENGTH OF RCVVAR,
                           FORMAT-NAME-1,
                           EXIT-POINT-NAME OF MISC,
                           FORMAT-NAME-2, EXIT-PGM-NBR,
                           NBR-OF-SELECT-CRITERIA, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                           OPEN OUTPUT LISTING,
                           MOVE EXCEPTION-ID OF QUS-EC
                             TO EXCEPTION-ID OF BAD-EXIT-PGM,
                           WRITE LIST-LINE FROM BAD-EXIT-PGM,
                           STOP RUN.
*
* If the call to retrieve exit program information is successful,
* check to see if there are any exit programs to call.
*
* The receiver variable offers enough room for a minimum of one
* exit program entry because the receiver variable was declared
* as 3500 bytes.  Therefore, this example only checks the
```

```
* number of exit programs returned field.  If the receiver
* variable were not large enough to hold at least one entry,
* the bytes available field would need to be checked as well as
* the number of exit programs returned field.  If the number of
* exit programs returned field is set to zero and the bytes
* available field is greater than the bytes returned field, the
* API had at least one exit program entry to return but was
* unable to because the receiver variable was too small.
*
     SET ADDRESS OF QUS-EXTI0200-ENTRY
         TO ADDRESS OF RCVVAR(OFFSET-PROGRAM-ENTRY
                                  OF QUS-EXTI0200 + 1:).
     PERFORM CALL-PGMS
             NUMBER-PROGRAMS-RETURNED OF QUS-EXTI0200 TIMES.
*
 CALL-PGMS.
*
* Call the exit program while ignoring failures on the call
*
     CALL PROGRAM-NAME OF QUS-EXTI0200-ENTRY USING
                                EXIT-PARAMETERS
                                ON EXCEPTION CONTINUE.
*
* Address the next exit program entry
*
     SET ADDRESS OF QUS-EXTI0200-ENTRY
             TO ADDRESS OF RCVVAR(OFFSET-NEXT-ENTRY
                                  OF QUS-EXTI0200-ENTRY + 1:).
```

Related tasks:

"Continuation handle" on page 76
When a call to an API is made and the API has more information to return than what can fit in the receiver variable or the user space, the API returns a continuation handle, which is used to mark the last value put in the receiver variable or the user space.

Related reference:

"Example in ILE C: Retrieving exit point and exit program information" on page 211
This ILE C program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

Example in OPM RPG: Retrieving exit point and exit program information

This OPM RPG program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

The Retrieve Exit Information (QUSRTVEI) API returns a continuation handle when it has more information to return than what can fit in the receiver variable.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*************************************************************
F*************************************************************
F*
F*  Program:      Retrieve Exit Point and Exit Program Information
F*
F*  Language:     OPM RPG
F*
F*  Description:  This program retrieves exit point and exit
F*                program information.  After retrieving the
F*                exit point information, the program calls each
F*                exit program.
F*
F*  APIs Used:    QUSRTVEI - Retrieve Exit Information
F*
F*************************************************************
```

```
F*************************************************************
F*
FQPRINT  O   F     132            PRINTER                      UC
I*
I* Error Code parameter include.  As this sample program
I* uses /COPY to include the error code structure, only the first
I* 16 bytes of the error code structure are available.  If the
I* application program needs to access the variable length
I* exception data for the error, the developer should physically
I* copy the QSYSINC include and modify the copied include to
I* define additional storage for the exception data.
I*
I/COPY QSYSINC/QRPGSRC,QUSEC
I*
I* Formats for the Retrieve Exit Information API.
I*
I/COPY QSYSINC/QRPGSRC,QUSREG
I*
I* Miscellaneous data
I*
I            DS
I I            'EXAMPLE_EXIT_POINT  '    1  20 EPNTNM
I I            -1                     B  21  240EPGMNB
I I            3500                   B  25  280RCVSZ
I                                     B  29  320X
I                                     B  33  360Y
I                                        37  57 CALLPG
IRCV         DS                          3500
C*
C* Beginning of mainline
C*
C* Retrieve the exit point information first.  If the current
C* number of exit programs is not zero, retrieve the exit
C* programs.  It is not necessary to call for the exit point
C* information to determine if the exit point has any exit
C* programs.  It is done here for illustrative purposes only.
C* You can make one call to the API for the exit program
C* information and check the number of exit program entries
C* returned field to see if there are any exit programs to call.
C*
C* Initialize the error code to inform the API that all
C* exceptions should be returned through the error code parameter.
C*
C                     Z-ADD16        QUSBNB
C*
C* Blank out the continuation handle to let the API know that this
C* is a first attempt at the retrieve operation.
C*
C                     MOVE *BLANKS   CONTHD 16
C*
C* Call the API to retrieve the exit point information
C*
C                     CALL 'QUSRTVEI'
C                     PARM           CONTHD
C                     PARM           RCV
C                     PARM           RCVSZ
C                     PARM 'EXTI0100'FORMAT  8
C                     PARM           EPNTNM
C                     PARM 'EXMP0100'EPTFMT  8
C                     PARM           EPGMNB
C                     PARM 0         QUSCCB
C                     PARM           QUSBN
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
```

```
C*
C           QUSBNC    IFGT 0
C                     OPEN QPRINT
C                     EXCPTERREPT
C                     EXSR DONE
C                     ENDIF
C*
C* If the call to retrieve exit point information is successful,
C* check to see if there are any exit programs to call.
C*
C           36        SUBSTRCV:1     QUSCG
C           QUSCGG    IFGT 0
C           1         ADD  QUSCGF    X
C           201       SUBSTRCV&#58;X     QUSCF
C           QUSCFF    IFGT 0
C*
C* There are some exit programs to call.  Blank out the continuation
C* handle to let the API know that this is a first attempt at the
C* retrieve operation.
C*
C                     MOVE *BLANKS   CONTHD
C*
C* Call the exit programs
C*
C                     EXSR CUSREI
C*
C* If the continuation handle field in the receiver variable is
C* not set to blanks, the API has more information to return than
C* what could fit in the receiver variable.  Call the API for
C* more exit programs to call.
C*
C           QUSCGD    DOWNE*BLANKS
C                     MOVELQUSCGD    CONTHD
C                     EXSR CUSREI
C                     ENDDO
C                     ENDIF
C                     ENDIF
C                     EXSR DONE
C*
C* End of MAINLINE
C*
C* Process exit programs in receiver variable
C*
C           CUSREI    BEGSR
C*
C* Call the API to retrieve the exit program information
C*
C                     CALL 'QUSRTVEI'
C                     PARM           CONTHD
C                     PARM           RCV
C                     PARM           RCVSZ
C                     PARM 'EXTI0200'FORMAT  8
C                     PARM           EPNTNM
C                     PARM 'EXMP0100'EPTFMT  8
C                     PARM           EPGMNB
C                     PARM 0         QUSCCB
C                     PARM           QUSBN
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C           QUSBNC    IFGT 0
C                     OPEN QPRINT
C                     EXCPTERRPGM
C                     EXSR DONE
```

```
C                     ENDIF
C*
C* If the call to retrieve exit program information is successful,
C* check to see if there are any exit programs to call.
C*
C* The receiver variable offers enough room for a minimum of one
C* exit program entry because the receiver variable was declared
C* as 3500 bytes.  Therefore, this example only checks the
C* number of exit programs returned field.  If the receiver
C* variable were not large enough to hold at least one entry,
C* the bytes available field would need to be checked as well as
C* the number of exit programs returned field.  If the number of
C* exit programs returned field is set to zero and the bytes
C* available field is greater than the bytes returned field, the
C* API had at least one exit program entry to return but was
C* unable to because the receiver variable was too small.
C*
C           36        SUBSTRCV:1     QUSCJ
C           1         ADD  QUSCJF    Y
C           72        SUBSTRCV&#58;Y     QUSCH
C                     DO   QUSCJG
C*
C* Get the exit program name and library
C*
C                     MOVE *BLANKS   CALLPG
C                     MOVELQUSCHL    CALLPG
C           CALLPG    CAT  '/':0     CALLPG
C           CALLPG    CAT  QUSCHK:0  CALLPG
C*
C* Call the exit program while ignoring failures on the call
C*
C                     CALL CALLPG                 01
C                     PARM           EXTPRM 10
C*
C* Set Y to point to the next exit program entry
C*
C           1         ADD  QUSCHB    Y
C           72        SUBSTRCV&#58;Y     QUSCH
C                     ENDDO
C                     ENDSR
C*
C* Return to programs caller
C           DONE      BEGSR
C                     SETON                     LR
C                     RETRN
C                     ENDSR
O*
OQPRINT  E  106           ERREPT
O                                       'Attempt to retrieve infor'
O                                       'mation failed: '
O                         QUSBND
OQPRINT  E  106           ERRPGM
O                                       'Attempt to retrieve Exit'
O                                       ' Programs failed: '
O                         QUSBND
```

Related tasks:

"Continuation handle" on page 76
When a call to an API is made and the API has more information to return than what can fit in the receiver variable or the user space, the API returns a continuation handle, which is used to mark the last value put in the receiver variable or the user space.

Related reference:

"Example in ILE C: Retrieving exit point and exit program information" on page 211
This ILE C program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

Example in ILE RPG: Retrieving exit point and exit program information

This ILE RPG program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

The Retrieve Exit Information (QusRetrieveExitInformation) API returns a continuation handle when it has more information to return than what can fit in the receiver variable.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*****************************************************************
F*****************************************************************
F*
F*  Program:      Retrieve Exit Point and Exit Program Information
F*
F*  Language:     ILE RPG
F*
F*  Description:  This program retrieves exit point and exit
F*                program information.  After retrieving the
F*                exit point information, the program calls each  ,
F*                exit program.
F*
F*  APIs Used:    QusRetrieveExitInformation - Retrieve Exit
F*                                             Information
F*
F*****************************************************************
F*****************************************************************
F*
FQPRINT    O    F  132        PRINTER OFLIND(*INOF) USROPN
D*
D* The following QUSREG include from QSYSINC is copied into
D* this program so that the data structures can be declared as
D* BASED.
D*
D*** START HEADER FILE SPECIFICATIONS ****************************
D*
D*Header File Name: H/QUSREG
D*
D*Descriptive Name: Standard Registration Structures.
D*
D*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
D*All rights reserved.
D*US Government Users Restricted Rights -
D*Use, duplication or disclosure restricted
D*by GSA ADP Schedule Contract with IBM Corp.
D*
D*Licensed Materials-Property of IBM
D*
D*
D*Description: All of the structures that are used in the
D*          Registration facilities are kept here to avoid
D*          conflict due to repetition.
D*
D*Header Files Included: None.
D*
D*Macros List: None.
D*
D*Structure List: Qus_Prep_Exit_t
D*             Qus_Qmff_t
D*             Qus_Selcrtr_t
D*             Qus_Select_Entry_t
D*             Qus_Program_Data_t
D*             Qus_EXTI0100_t
D*             Qus_EXTI0100_Entry_t
D*             Qus_EXTI0200_t
D*             Qus_EXTI0200_Entry_t
```

```
D*             Qus_EXTI0300_t
D*             Qus_EXTI0300_Entry_t
D*
D*Function Prototype List: none.
D*
D*Change Activity:
D*
D*CFD List:
D*
D*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
D*---- ------------ ----- ------ --------- ----------------------
D*$A0= D2862000     3D10  940327 LUPA:     New Include
D*
D*End CFD List.
D*
D*Additional notes about the Change Activity
D*End Change Activity.
D*** END HEADER FILE SPECIFICATIONS *****************************
D*****************************************************************
D*Format structure for the Preprocessing Exit Program Format for
D*QusRegisterExitPoint API.
D*****************************************************************
DQUSPE            DS
D*                                          Qus Prep Exit
D QUSPPN                  1     10
D*                                          Prep Prog Name
D QUSPPLIB               11     20
D*                                          Prep Prog Library
D QUSPPF                 21     28
D*                                          Prep Prog Format
D*****************************************************************
D*Format structure for the Qualified Message File Format for the
D*entire service program.
D*****************************************************************
DQUSQMFF          DS
D*                                          Qus Qmff
D QUSMFIL                 1     10
D*                                          Message File
D QUSMLIB                11     20
D*                                          Message Library
D QUSMI                  21     27
D*                                          Message Id
D*****************************************************************
D*Format structure for the Exit Program Selection Criteria of the
D*QusRetrieveExitInformation API.
D****                                                          ***
D*NOTE: This structure only defines fixed fields. Any varying
D*   length or repeating field will have to be defined by
D*   the user.
D*****************************************************************
DQUSSE            DS
D*                                          Qus Select Entry
D QUSSE00                 1      4B 0
D*                                          Size Entry
D QUSCO                   5      8B 0
D*                                          Comp Operator
D QUSSPD                  9     12B 0
D*                                          Start Pgm Data
D QUSLCD                 13     16B 0
D*                                          Length Comp Data
D*QUSCD                  17     17
D*
D*                                    Varying length
DQUSS             DS
D*                                          Qus Selcrtr
D QUSNBRSC                1      4B 0
D*                                          Number Sel Criteria
```

```
D*QUSARRAY                      17    DIM(00001)
D* QUSSE01                       9B 0 OVERLAY(QUSARRAY:00001)
D* QUSCO00                       9B 0 OVERLAY(QUSARRAY:00005)
D* QUSSPD00                      9B 0 OVERLAY(QUSARRAY:00009)
D* QUSLCD00                      9B 0 OVERLAY(QUSARRAY:00013)
D* QUSCD00                       1    OVERLAY(QUSARRAY:00017)
D*
D*                                    Varying length
D*****************************************************************
D*Format Structure for the Program Data.  This structure has
D*set up to facilitate COBOL and RPG pointer basing.
D*****************************************************************
DQUSPGMD          DS
D*                                              Qus Program Data
D QUSDATA01               1      1
D*                                    Varying length
D*****************************************************************
D*Format structure for the EXTI0100 Format for the
D*QusRetrieveExitInformation API.
D****                                                          ***
D*NOTE: This structure only defines fixed fields. Any varying
D*   length or repeating field will have to be defined by
D*   the user.
D*****************************************************************
DQUS0100E         DS                  BASED(INFSPCPTR)
D*                                              Qus EXTI0100 Entry
D QUSEPN00                1     20
D*                                              Exit Point Name
D QUSFN08                21     28
D*                                              Format Name
D QUSMEP                 29     32B 0
D*                                              Max Exit Programs
D QUSNBREP               33     36B 0
D*                                              Number Exit Programs
D QUSAD                  37     37
D*                                              Allow Deregistration
D QUSACC                 38     38
D*                                              Allow Change Control
D QUSREP                 39     39
D*                                              Registered Exit Point
D QUSPNAP                40     49
D*                                              Prep Name Add Pgm
D QUSPLAP                50     59
D*                                              Prep Lib Add Pgm
D QUSPFA                 60     67
D*                                              Prep Format Add
D QUSPNRP                68     77
D*                                              Prep Name Rmv Pgm
D QUSPLRP                78     87
D*                                              Prep Lib Rmv Pgm
D QUSPFR                 88     95
D*                                              Prep Format Rmv
D QUSPNRI                96    105
D*                                              Prep Name Rtv Info
D QUSPLRI               106    115
D*                                              Prep Lib Rtv Info
D QUSPFR00              116    123
D*                                              Prep Format Rtv
D QUSDI                 124    124
D*                                              Desc Indicator
D QUSDMFIL              125    134
D*                                              Desc Msg File
D QUSDMLIB              135    144
D*                                              Desc Msg Library
D QUSDMI                145    151
D*                                              Desc Msg Id
D QUSTD                 152    201
```

```
D*                                          Text Description
D*QUSERVED03            202    202
D*
D*                                    Varying length
DQUSI0100         DS                BASED(BASSPCPTR)
D*                                          Qus EXTI0100
D QUSBRTN                1      4B 0
D*                                          Bytes Returned
D QUSBAVL00              5      8B 0
D*                                          Bytes Available
D QUSCH                  9     24
D*                                          Continue Handle
D QUSOEPE               25     28B 0
D*                                          Offset Exit Point Entry
D QUSNBRPR              29     32B 0
D*                                          Number Points Returned
D QUSLEPE               33     36B 0
D*                                          Length Exit Point Entry
D*QUSERVED04            37     37
D*
D*                                    Varying length
D*QUSARRAY00                  202    DIM(00001)
D* QUSEPN01                     20    OVERLAY(QUSARRAY00:00001)
D* QUSFN09                      8    OVERLAY(QUSARRAY00:00021)
D* QUSMEP00                     9B 0 OVERLAY(QUSARRAY00:00029)
D* QUSNBREP00                   9B 0 OVERLAY(QUSARRAY00:00033)
D* QUSAD00                      1    OVERLAY(QUSARRAY00:00037)
D* QUSACC00                     1    OVERLAY(QUSARRAY00:00038)
D* QUSREP00                     1    OVERLAY(QUSARRAY00:00039)
D* QUSPNAP00                   10    OVERLAY(QUSARRAY00:00040)
D* QUSPLAP00                   10    OVERLAY(QUSARRAY00:00050)
D* QUSPFA00                    8    OVERLAY(QUSARRAY00:00060)
D* QUSPNRP00                   10    OVERLAY(QUSARRAY00:00068)
D* QUSPLRP00                   10    OVERLAY(QUSARRAY00:00078)
D* QUSPFR01                    8    OVERLAY(QUSARRAY00:00088)
D* QUSPNRI00                   10    OVERLAY(QUSARRAY00:00096)
D* QUSPLRI00                   10    OVERLAY(QUSARRAY00:00106)
D* QUSPFR02                    8    OVERLAY(QUSARRAY00:00116)
D* QUSDI00                     1    OVERLAY(QUSARRAY00:00124)
D* QUSDMFIL00                  10    OVERLAY(QUSARRAY00:00125)
D* QUSDMLIB00                  10    OVERLAY(QUSARRAY00:00135)
D* QUSDMI00                    7    OVERLAY(QUSARRAY00:00145)
D* QUSTD00                     50    OVERLAY(QUSARRAY00:00152)
D* QUSERVED05                  1    OVERLAY(QUSARRAY00:00202)
D*
D*                                    Varying length
D*****************************************************************
D*Format structure for the EXTI0200 Format for the
D*QusRetrieveExitInformation API.
D****                                                          ***
D*NOTE: This structure only defines fixed fields. Any varying
D*   length or repeating field will have to be defined by
D*   the user.
D*****************************************************************
DQUS0200E         DS                BASED(INFSPCPTR)
D*                                          Qus EXTI0200 Entry
D QUSONE                 1      4B 0
D*                                          Offset Next Entry
D QUSEPN02               5     24
D*                                          Exit Point Name
D QUSFN10               25     32
D*                                          Format Name
D QUSREP01              33     33
D*                                          Registered Exit Pt
D QUSCE                 34     34
D*                                          Complete Entry
D QUSERVED06            35     36
```

```
D*                                              Reserved
D QUSPGMN                37     40B 0
D*                                              Program Number
D QUSPGMN00              41     50
D*                                              Program Name
D QUSPGML                51     60
D*                                              Program Library
D QUSDC                  61     64B 0
D*                                              Data CCSID
D QUSOED                 65     68B 0
D*                                              Offset Exit Data
D QUSLED                 69     72B 0
D*                                              Length Exit Data
D*QUSERVED06             73     73
D*
D*                                      Varying length
D*QUSPD                   1
D* QUSDATA02              74     74
D*
D*                                      Varying length
DQUSI0200         DS                  BASED(BASSPCPTR)
D*                                              Qus EXTI0200
D QUSBRTN00               1      4B 0
D*                                              Bytes Returned
D QUSBAVL01               5      8B 0
D*                                              Bytes Available
D QUSCH00                 9     24
D*                                              Continue Handle
D QUSOPGME               25     28B 0
D*                                              Offset Program Entry
D QUSNBRPR00             29     32B 0
D*                                              Number Programs Returned
D QUSLPGME               33     36B 0
D*                                              Length Program Entry
D*QUSERVED07             37     37
D*
D*                                      Varying length
D*QUSARRAY01                    74    DIM(00001)
D* QUSONE00                      9B 0 OVERLAY(QUSARRAY01:00001)
D* QUSEPN03                     20    OVERLAY(QUSARRAY01:00005)
D* QUSFN11                       8    OVERLAY(QUSARRAY01:00025)
D* QUSREP02                      1    OVERLAY(QUSARRAY01:00033)
D* QUSCE00                       1    OVERLAY(QUSARRAY01:00034)
D* QUSERVED08                    2    OVERLAY(QUSARRAY01:00035)
D* QUSPGMN01                    9B 0 OVERLAY(QUSARRAY01:00037)
D* QUSPGMN02                   10    OVERLAY(QUSARRAY01:00041)
D* QUSPGML00                   10    OVERLAY(QUSARRAY01:00051)
D* QUSDC00                      9B 0 OVERLAY(QUSARRAY01:00061)
D* QUSOED00                     9B 0 OVERLAY(QUSARRAY01:00065)
D* QUSLED00                     9B 0 OVERLAY(QUSARRAY01:00069)
D* QUSERVED08                   1    OVERLAY(QUSARRAY01:00073)
D* QUSPD00                      1
D*  QUSDATA03                  1    OVERLAY(QUSARRAY01:00001)
D*
D*                                      Varying length
D*****************************************************************
D*Format structure for the EXTI0300 Format for the
D*QusRetrieveExitInformation API.
D****                                                          ***
D*NOTE: This structure only defines fixed fields. Any varying
D*   length or repeating field will have to be defined by
D*   the user.
D*****************************************************************
DQUS0300E         DS
D*                                              Qus EXTI0300 Entry
D QUSONE01                1      4B 0
D*                                              Offset Next Entry
```

```
D QUSEPN04                5     24
D*                                          Exit Point Name
D QUSFN12                25     32
D*                                          Format Name
D QUSREP03               33     33
D*                                          Registered Exit Point
D QUSCE01                34     34
D*                                          Complete Entry
D QUSERVED09             35     36
D*                                          Reserved
D QUSPGMN03              37     40B 0
D*                                          Program Number
D QUSPGMN04              41     50
D*                                          Program Name
D QUSPGML01              51     60
D*                                          Program Library
D QUSDI01                61     61
D*                                          Desc Indicator
D QUSMFIL00              62     71
D*                                          Message File
D QUSMFILL               72     81
D*                                          Message File Library
D QUSMI00                82     88
D*                                          Message Id
D QUSTD01                89    138
D*                                          Text Desc
D QUSRSV201             139    140
D*                                          Reserved2
D QUSDC01               141    144B 0
D*                                          Data CCSID
D QUSOPD                145    148B 0
D*                                          Offset Pgm Data
D QUSLPD                149    152B 0
D*                                          Length Pgm Data
D*QUSERVED09            153    153
D*
D*                                  Varying length
D*QUSPD01                 1
D* QUSDATA04             154    154
D*
D*                                  Varying length
DQUSI0300         DS
D*                                          Qus EXTI0300
D QUSBRTN01               1      4B 0
D*                                          Bytes Returned
D QUSBAVL02               5      8B 0
D*                                          Bytes Available
D QUSCH01                 9     24
D*                                          Continue Handle
D QUSOPGME00             25     28B 0
D*                                          Offset Program Entry
D QUSNBRPR01             29     32B 0
D*                                          Number Programs Returned
D QUSLPGME00             33     36B 0
D*                                          Length Program Entry
D*QUSERVED10             37     37
D*
D*                                  Varying length
D*QUSARRAY02                   154    DIM(00001)
D* QUSONE02                     9B 0 OVERLAY(QUSARRAY02:00001)
D* QUSEPN05                    20    OVERLAY(QUSARRAY02:00005)
D* QUSFN13                     8    OVERLAY(QUSARRAY02:00025)
D* QUSREP04                    1    OVERLAY(QUSARRAY02:00033)
D* QUSCE02                     1    OVERLAY(QUSARRAY02:00034)
D* QUSERVED11                  2    OVERLAY(QUSARRAY02:00035)
D* QUSPGMN05                   9B 0 OVERLAY(QUSARRAY02:00037)
D* QUSPGMN06                  10    OVERLAY(QUSARRAY02:00041)
```

```
D* QUSPGML02                    10    OVERLAY(QUSARRAY02:00051)
D* QUSDI02                       1    OVERLAY(QUSARRAY02:00061)
D* QUSMFIL01                    10    OVERLAY(QUSARRAY02:00062)
D* QUSMFILL00                   10    OVERLAY(QUSARRAY02:00072)
D* QUSMI01                       7    OVERLAY(QUSARRAY02:00082)
D* QUSTD02                      50    OVERLAY(QUSARRAY02:00089)
D* QUSRSV202                     2    OVERLAY(QUSARRAY02:00139)
D* QUSDC02                       9B 0 OVERLAY(QUSARRAY02:00141)
D* QUSOPD00                      9B 0 OVERLAY(QUSARRAY02:00145)
D* QUSLPD00                      9B 0 OVERLAY(QUSARRAY02:00149)
D* QUSERVED11                    1    OVERLAY(QUSARRAY02:00153)
D* QUSPD02                       1
D*  QUSDATA05                    1    OVERLAY(QUSARRAY02:00001)
D*
D*                                      Varying length
D*
D* Error Code parameter include.  As this sample program
D* uses /COPY to include the error code structure, only the first
D* 16 bytes of the error code structure are available.  If the
D* application program needs to access the variable length
D* exception data for the error, the developer should physically
D* copy the QSYSINC include and modify the copied include to
D* define additional storage for the exception data.
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D*****************************************************************
D*Prototype for calling Retrieve Exit Information
D*****************************************************************
D QUSREI          C                   'QusRetrieveExitInformation'
D*
D* Miscellaneous data
D*
DEPNTNAME         S             20    INZ('EXAMPLE_EXIT_POINT')
DEPGM_NBR         S              9B 0 INZ(-1)
DRCVVAR           S              1    DIM(3500)
DRCVVAR_SZ        S              9B 0 INZ(%SIZE(RCVVAR:*ALL))
DBASSPCPTR        S               *
DINFSPCPTR        S               *
DCALL_PGM         S             21
C*
C* Beginning of mainline
C*
C* Retrieve the exit point information first.  If the current
C* number of exit programs is not zero, retrieve the exit
C* programs.  It is not necessary to call for the exit point
C* information to determine if the exit point has any exit
C* programs.  It is done here for illustrative purposes only.
C* You can make one call to the API for the exit program
C* information and check the number of exit program entries
C* returned field to see if there are any exit programs to call.
C*
C* Initialize the error code to inform the API that all
C* exceptions should be returned through the error code parameter.
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Blank out the continuation handle to let the API know that this
C* is a first attempt at the retrieve operation.
C*
C                   MOVE      *BLANKS       CONTIN_HDL       16
C*
C* Call the API to retrieve the exit programs
C*
C                   CALLB     QUSREI
C                   PARM                    CONTIN_HDL
C                   PARM                    RCVVAR
```

```
C                   PARM                    RCVVAR_SZ
C                   PARM      'EXTI0100'    FORMAT            8
C                   PARM                    EPNTNAME
C                   PARM      'EXMP0100'    EPNT_FMT          8
C                   PARM                    EPGM_NBR
C                   PARM      0             QUSNBRSC
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   OPEN      QPRINT
C                   EXCEPT    ERRAEPNT
C                   EXSR      DONE
C                   ENDIF
C*
C* If the call to retrieve exit point information is successful,
C* check to see if there are any exit programs to call.
C*
C                   EVAL      BASSPCPTR = %ADDR(RCVVAR)
C                   IF        QUSNBRPR > 0
C                   EVAL      INFSPCPTR = %ADDR(RCVVAR(QUSOEPE+1))
C                   IF        QUSNBREP > 0
C*
C* There are some exit programs to call.  Blank out the continuation
C* handle to let the API know that this is a first attempt at the
C* retrieve operation.
C*
C                   EVAL      CONTIN_HDL = *BLANKS
C*
C* Call the exit programs
C*
C                   EXSR      CUSREI
C*
C* If the continuation handle field in the receiver variable is
C* not set to blanks, the API has more information to return than
C* what could fit in the receiver variable.  Call the API for
C* more exit programs to call.
C*
C                   DOW       QUSCH00 <> *BLANKS
C                   EVAL      CONTIN_HDL = QUSCH00
C                   EXSR      CUSREI
C                   ENDDO
C                   ENDIF
C                   ENDIF
C                   EXSR      DONE
C*
C* End of MAINLINE
C*
C* Process exit programs in receiver variable
C*
C     CUSREI        BEGSR
C*
C* Call the API to retrieve the exit program information
C*
C                   CALLB     QUSREI
C                   PARM                    CONTIN_HDL
C                   PARM                    RCVVAR
C                   PARM                    RCVVAR_SZ
C                   PARM      'EXTI0200'    FORMAT            8
C                   PARM                    EPNTNAME
C                   PARM      'EXMP0100'    EPNT_FMT          8
C                   PARM                    EPGM_NBR
C                   PARM      0             QUSNBRSC
```

```
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   OPEN      QPRINT
C                   EXCEPT    ERRAEPGM
C                   EXSR      DONE
C                   ENDIF
C*
C* If the call to retrieve exit program information is successful,
C* check to see if there are any exit programs to call.
C*
C* The receiver variable offers enough room for a minimum of one
C* exit program entry because the receiver variable was declared
C* as 3500 bytes.  Therefore, this example only checks the
C* number of exit programs returned field.  If the receiver
C* variable were not large enough to hold at least one entry,
C* the bytes available field would need to be checked as well as
C* the number of exit programs returned field.  If the number of
C* exit programs returned field is set to zero and the bytes
C* available field is greater than the bytes returned field, the
C* API had at least one exit program entry to return but was
C* unable to because the receiver variable was too small.
C*
C                   EVAL      INFSPCPTR = %ADDR(RCVVAR(QUSOPGME+1))
C                   DO        QUSNBRPR00
C*
C* Get the exit program name and library
C*
C                   EVAL      CALL_PGM = %TRIMR(QUSPGML) +
C                                        '/' + QUSPGMN00
C*
C* Call the exit program while ignoring failures on the call
C*
C                   CALL      CALL_PGM                            01
C                   PARM                    EXIT_PARMS       10
C*
C* Set INFSPCPTR to point to the next exit program entry
C*
C                   EVAL      INFSPCPTR = %ADDR(RCVVAR(QUSONE+1))
C                   ENDDO
C                   ENDSR
C*
C* Return to programs caller
C     DONE          BEGSR
C                   EVAL      *INLR = '1'
C                   RETURN
C                   ENDSR
O*
OQPRINT    E            ERRAEPNT       1  6
O                                              'Attempt to retrieve infor-
O                                              mation failed: '
O                       QUSEI
OQPRINT    E            ERRAEPGM       1  6
O                                              'Attempt to retrieve Exit -
O                                              Programs failed: '
O                       QUSEI
```

Related tasks:

"Continuation handle" on page 76
When a call to an API is made and the API has more information to return than what can fit in the receiver variable or the user space, the API returns a continuation handle, which is used to mark the last value put in the receiver variable or the user space.

Related reference:

"Example in ILE C: Retrieving exit point and exit program information" on page 211
This ILE C program retrieves exit point and exit program information. It then resolves to each exit program and calls the exit program.

Performing tasks using APIs

You can use APIs to perform different types of tasks.

Related reference:

"Examples: APIs and exit programs" on page 297
These examples show how to use a wide variety of APIs and exit programs.

Examples: Packaging your own software products

You can define, create, distribute, and maintain your own software product using APIs. These examples show how to use the APIs to package a product similar to the way IBM does.

Creating the example product:

To package your product, you first create all the objects that comprise your product.

The first example product being packaged is called ABC Product. The product is made up of one library, ABC, with no options off of this product. ABC Product consists of the following objects.

Table 19. ABC software packaging

Number	Object name	Object type	Text description
1	ABCPGMMRM1	*PGM	MRM [1] preprocessing program
2	ABCPGMMRM2	*PGM	MRM postprocessing program
3	ABCPGMMRI1	*PGM	MRI [2] preprocessing program
4	ABCPGMMRI2	*PGM	MRI postprocessing program
5	ABCPGM	*PGM	CPP [3] for ABC command
6	QCLSRC	*FILE(SRCPF)	Source physical file
7	ABCDSPF	*FILE(DSPF)	Display file
8	ABCPF	*FILE(PF)	Physical file
9	ABCMSG	*MSGF	Message file
10	ABC	*CMD	Command for ABC Product
11	ABCPNLGRP	*PNLGRP	Panels for ABC
12	ABC0050	*PRDDFN	Product definition
13	ABC0029	*PRDLOD	Product load for MRI
14	ABC0050	*PRDLOD	Product load for MRM
15	ABC	*LIB	ABC Product

Notes:

1. Machine readable material
2. Machine readable information
3. Command processing program

You create all the objects (numbers 1 through 11 and number 15 in the table) that comprise your product. "Example in CL: Creating objects for packaging a product" shows the code that creates the objects. After you create the objects, you follow the steps listed in "Example in ILE COBOL: Packaging a product" on page 253.

The following figure is an overview of the steps required to create a product. An explanation is given in the figure below of the numbers. The same numbers also appear in the code.

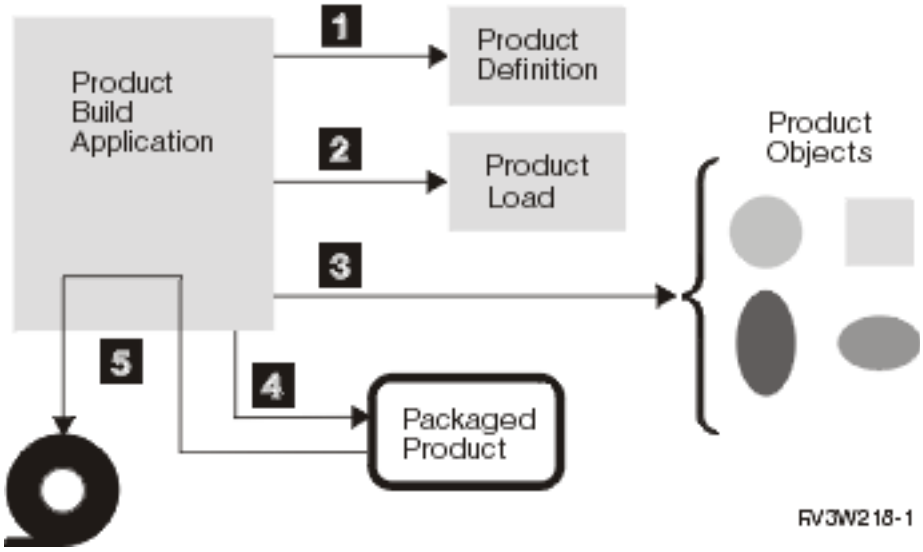


Figure 2. Steps for creating a software product

(1) Create a product definition with information about the licensed program, such as ID, version, and release.

(2) Create a product load, which further defines each option of a licensed program, such as the libraries, folders, and exit programs that comprise the product.

(3) Identify all objects associated with the product by changing the product ID, release level, product option, and load ID in the object description by using the Change Object Description API.

(4) Package the product. Verify and store a list of all objects marked for this product in the product load object.

(5) Use the Save Licensed Program (SAVLICPGM) command to save the product to tape.

Example in CL: Creating objects for packaging a product:

This CL program creates objects 1 through 11 and 15 that comprise your product.

Objects 1 through 11 and 15 are listed in Table 19 on page 238.

```
PGM
/* Delete library and start from scratch */
  DLTLIB ABC

/* MRM Objects */
  CRTLIB ABC
  CRTCLPGM ABC/ABCPGMMRM1  ABCDEV/QCLSRC +
           TEXT('MRM Preprocessing Program')
  CRTCLPGM ABC/ABCPGMMRM2  ABCDEV/QCLSRC +
           TEXT('MRM Postprocessing Program')
  CRTCLPGM ABC/ABCPGM      ABCDEV/QCLSRC +
           TEXT('CPP for ABC command')
```

```
/* MRI Objects */
  CRTCLPGM ABC/ABCPGMMRI1  ABCDEV/QCLSRC +
           TEXT('MRI Preprocessing Program')
  CRTCLPGM ABC/ABCPGMMRI2  ABCDEV/QCLSRC +
           TEXT('MRI Postprocessing Program')
  CRTSRCPF ABC/QCLSRC TEXT('Source Physical File for ABC Product')
  CRTDSPF ABC/ABCDSPF ABCDEV/QDDSSRC +
           TEXT('Display File for ABC Product')
  CRTPF ABC/ABCPF ABCDEV/QDDSSRC +
           TEXT('Physical File for ABC Product')
  CRTMSGF ABC/ABCMSG TEXT('Message File')
  ADDMSGD ABC0001 ABC/ABCMSG MSG('ABC Product')
  CRTCMD ABC/ABC ABC/ABCPGM ABCDEV/QCMDSRC +
           TEXT('Command for ABC Product')
  CRTPNLGRP ABC/ABCPNLGRP ABCDEV/QPNLSRC +
           TEXT('Panel for ABC Command')

/* The next program creates the product definitions, product loads, */
/* and gives all the objects associated with the product the correct*/
/* product information.  It packages the product, which enables     */
/* you to use the SAVLICPGM, RSTLICPGM, and DLTLICPGM commands.     */

  CRTRPGPGM ABCDEV/SFTWPRDEX ABCDEV/QRPGSRC
/*  (1)  (2)  (3)  (4)                                              */
  CALL ABCDEV/SFTWPRDEX
ENDPGM
```

Example in OPM RPG: Packaging a product:

This OPM RPG program creates objects 12 through 14 and packages your product.

Objects 12 through 14 are listed in Table 19 on page 238.

```
     F*****************************************************************
     F*****************************************************************
     F*
     F*Program Name: SFTWPRDEX
     F*
     F*Language:  OPM RPG
     F*
     F*Descriptive Name: Software Product Example
     F*
     F*Description: This example contains the steps necessary to
     F*             package your product like IBM products.
     F*
     F*Header Files Included: QUSEC - Error Code Parameter
     F*                              (Copied into Program)
     F*                       QSZCRTPD - Create Product Definition API
     F*                       QSZCRTPL - Create Product Load API
     F*                       QSZPKGPO - Package Product Option API
     F*
     F*****************************************************************
     F*****************************************************************
     FQPRINT  O   F     132     OF     PRINTER
     E* COMPILE TIME ARRAY
     E                    OBJ   001  15 41
     I*
     I* Error Code Parameter Include for the APIs
     I*
     I* The following QUSEC include has been copied into this program
     I* so that the variable length field can be defined as a fixed
     I* length.
     I*** START HEADER FILE SPECIFICATIONS ****************************
     I*
     I*Header File Name: H/QUSEC
     I*
     I*Descriptive Name: Error Code Parameter.
```

```
I*
I*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
I*All rights reserved.
I*US Government Users Restricted Rights -
I*Use, duplication or disclosure restricted
I*by GSA ADP Schedule Contract with IBM Corp.
I*
I*Licensed Materials-Property of IBM
I*
I*
I*Description: Include header file for the error code parameter.
I*
I*Header Files Included: None.
I*
I*Macros List: None.
I*
I*Structure List: Qus_EC_t
I*
I*Function Prototype List: None.
I*
I*Change Activity:
I*
I*CFD List:
I*
I*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
I*---- ------------ ----- ------ --------- ----------------------
I*$A0= D2862000     3D10  931201 DPOHLSON: New Include
I*
I*End CFD List.
I*
I*Additional notes about the Change Activity
I*End Change Activity.
I*** END HEADER FILE SPECIFICATIONS *****************************
I*****************************************************************
I*Record structure for error code parameter
I****                                                          ***
I*NOTE: The following type definition only defines the fixed
I*   portion of the format.  Varying length field exception
I*   data will not be defined here.
I*****************************************************************
IQUSBN       DS
I*                                          Qus EC
I                                    B   1   40QUSBNB
I*                                          Bytes Provided
I                                    B   5   80QUSBNC
I*                                          Bytes Available
I                                        9  15 QUSBND
I*                                          Exception Id
I                                       16  16 QUSBNF
I*                                          Reserved
I*                                      17  17 QUSBNG
I*
I*                                      Varying length
I                                       17 100 QUSBNG
I*
I* Create Product Definition API Include
I*
I/COPY QSYSINC/QRPGSRC,QSZCRTPD
I*
I* Create Product Load API Include
I*
I/COPY QSYSINC/QRPGSRC,QSZCRTPL
I*
I* Package Product Option API Include
I*
I/COPY QSYSINC/QRPGSRC,QSZPKGPO
I*
```

```
I*
I            DS
I I            1                    B   1   40NUMPOP
I I            1                    B   5   80NUMLAN
I I            'ABC0050   ABC       '    9  28 PDFN
I I            'ABC Product'             29  78 TEXTD
I I            '5072535010    '          79  92 PHONE
I I            '*NODYNNAM '              93 102 ALWDYN
I I            '*USE      '             103 112 PUBAUT
I I            'ABCPGMMRM2'             113 122 POSTM
I I            'ABCPGMMRM1'             123 132 PREM
I I            'ABCPGMMRI2'             133 142 POSTI
I I            'ABCPGMMRI1'             143 152 PREI
I*
I* Change Object Information Parameter
ICOBJI       DS                             49
I I            3                    B   1   40NUMKEY
I I            13                   B   5   80KEY13
I I            4                    B   9  120LEN13
I                                      13  16 PID13
I I            12                   B  17  200KEY12
I I            4                    B  21  240LEN12
I                                      25  28 LID12
I I            5                    B  29  320KEY5
I I            13                   B  33  360LEN5
I                                      37  49 LP5
I*
I* Object Data Structure - Breakdown of fields in Array OBJ
IOBJDS       DS
I                                       1  10 NAME
I                                      11  20 TYP
I                                      21  24 PID
I                                      25  28 LID
I                                      29  41 LP
I            DS
I                                    B   1   40RCVLEN
I I            0                    B   5   80NUMBK
I I            1                    B   9  120NUMBL
I I            0                    B  13  160NUMBM
C*
C* Beginning of Mainline
C*
C* Create Product Definition Object - ABC0050
C*
C                     EXSR PRDDFN    (1)
C*
C* Create Product Load Objects - ABC0050 (MRM) and ABC0029 (MRI)
C*
C                     EXSR PRDLOD    (2)
C*
C* Change Object Description for all objects associated with
C* the ABC Product.
C*
C                     EXSR COBJD     (3)
C*
C* Package the ABC Product so that all the SAVLICPGM, RSTLIBPGM,
C* and DLTLICPGM commands work with the product.
C*
C                     EXSR PKGPO     (4)
C*
C* Complete; product is ready to ship.
C*
C                     SETON                    LR
C                     RETRN
C*
C* End of MAINLINE
C*
```

```
C*
C*****************************************************************
C*****************************************************************
C*
C* Subroutine: PRDDFN
C*
C* Descriptive Name: Create product definitions.
C*
C* Description: This subroutine creates the product definition
C*              ABC0050 for the ABC Product.
C*
C*****************************************************************
C*****************************************************************
C*
C           PRDDFN    BEGSR
C* Setup for Product Definition
C* Fill Product Definition Information Parameter
C                     Z-ADD100       QUSBNB
C                     MOVEL'0ABCABC' QSZBCB           Product ID
C                     MOVEL'V3R1M0'  QSZBCC           Release Level
C                     MOVEL'ABCMSG'  QSZBCD           Message File
C                     MOVEL'*CURRENT'QSZBCF           First Copyright
C                     MOVEL'*CURRENT'QSZBCG           Current Copyright
C                     MOVEL'941201'  QSZBCH           Release Date
C                     MOVEL'*NO'     QSZBCJ           Allow multiple rel.
C                     MOVEL'*PHONE'  QSZBCK           Registration ID Value
C                     MOVELPHONE     QSZBCL           Registration ID Value
C* Fill Product Load Parameter
C                     MOVEL'0000'    QSZBDB           Product Option Number
C                     MOVEL'ABC0001' QSZBDC           Message ID
C                     MOVELALWDYN    QSZBDD           Allow Dynamic Naming
C                     MOVEL'5001'    QSZBDF           Code Load ID
C                     MOVEL*BLANKS   QSZBDG           Reserved
C* Fill Language Load List Parameter
C                     MOVEL'2924    'QSZBFB           Language Load ID
C                     MOVEL'0000'    QSZBFC           Product Option Number
C                     MOVEL*BLANKS   QSZBFD           Reserved
C*
C* Create the Product Definition for the ABC Product
C*
C                     MOVEL'QSZCRTPD'API    10
C                     CALL 'QSZCRTPD'
C                     PARM           PDFN             Qual. Prod. Defn.
C                     PARM           QSZBC            Prod. Defn. Info.
C                     PARM           QSZBD            Prod. Option List
C                     PARM           NUMPOP           # Prod. Options
C                     PARM           QSZBF            Lang. Load List
C                     PARM           NUMLAN           # Lang. Load List
C                     PARM           TEXTD            Text Description
C                     PARM           PUBAUT           Public Authority
C                     PARM           QUSBN            Error Code
C* Check for errors returned in the error code parameter.
C                     EXSR ERRCOD
C                     ENDSR
C*
C*****************************************************************
C*****************************************************************
C*
C* Subroutine: PRDLOD
C*
C* Descriptive Name: Create product loads.
C*
C* Description: This subroutine creates the product loads,
C*              ABC0050 and ABC0029, for the ABC Product.
C*
C*****************************************************************
C*****************************************************************
```

```
C*
C           PRDLOD    BEGSR
C*
C* Setup for Product Load for MRM Objects
C* Fill Product Load Information Parameter
C                     MOVEL'0ABCABC' QSZBHB           Product ID
C                     MOVEL'V3R1M0'  QSZBHC           Release Level
C                     MOVEL'0000'    QSZBHD           Product Option
C                     MOVEL'*CODE'   QSZBHF           Product Load Type
C                     MOVEL'*CODEDFT'QSZBHG           Load ID
C                     MOVEL'*PRDDFN' QSZBHH           Registration ID Type
C                     MOVEL*BLANKS   QSZBHJ           Registration ID Value
C                     MOVEL'*CURRENT'QSZBCK           Min. Target Release
C                     MOVEL*BLANKS   QSZBCL           Reserved
C*
C* Fill Principal Library Information Parameter
C                     MOVEL'ABC'     QSZBJB           Prin. Dev. Lib. Name
C                     MOVEL'ABC'     QSZBJC           Prin. Prim. Lib. Name
C                     MOVELPOSTM     QSZBJD           Post-Exit Prog. Name
C*
C* Fill Preoperation Exit Programs Parameter
C                     MOVELPREM      QSZBLB           Pre-Exit Prog. Name
C                     MOVEL'ABC'     QSZBLC           Dev. Lib. Name
C*
C* Fill Additional Library List Parameter
C*       None
C*
C* Fill Folder List Parameter
C*       None
C*
C* Create the product load for the ABC Product - MRM Objects
C*
C                     MOVEL'QSZCRTPL'API
C                     CALL 'QSZCRTPL'
C                     PARM 'ABC0050' PRDIDN 10        Prod. ID Name
C                     PARM           QSZBH            Prod. Defn. Info.
C                     PARM *BLANKS   SECLIB 10        Sec. Lang. Lib
C                     PARM           QSZBJ            Principal Lib Info
C                     PARM           QSZBK            Add. Library List
C                     PARM 0         NUMBK            # Add. Lib. List
C                     PARM           QSZBL            Pre-Exit Programs
C                     PARM 1         NUMBL            # Pre-Exit Programs
C                     PARM           QSZBM            Folder List
C                     PARM 0         NUMBM            # Folder List
C                     PARM           TEXTD            Text Description
C                     PARM '*USE'    PUBAUT           Public Authority
C                     PARM           QUSBN            Error Code
C* Check for errors returned in the error code parameter.
C                     EXSR ERRCOD
C*
C* Setup for Product Load for MRI Objects
C* Fill Product Load Information Parameter
C                     MOVEL'*LNG '   QSZBHF           Product Load Type
C                     MOVEL'2924    'QSZBHG           Load ID
C*
C* Fill Principal Library Information Parameter
C                     MOVELPOSTI     QSZBJD           Post-Exit Prog. Name
C*
C* Fill Preoperation Exit Programs Parameter
C                     MOVELPREI      QSZBLB           Pre-Exit Prog. Name
C*
C* Fill Additional Library List Parameter
C*       None
C*
C* Fill Folder List Parameter
C*       None
C*
```

```
C* Create the product load for the ABC Product - MRI Objects
C*
C                     MOVEL'QSZCRTPL'API
C                     CALL 'QSZCRTPL'
C                     PARM 'ABC0029' PRDIDN 10        Prod. ID Name
C                     PARM           QSZBH            Prod. Defn. Info.
C                     PARM 'ABC2924 'SECLIB           Sec. Lang. Lib
C                     PARM           QSZBJ            Principal Lib Info
C                     PARM           QSZBK            Add. Library List
C                     PARM 0         NUMBK            # Add. Lib. List
C                     PARM           QSZBL            Pre-Exit Programs
C                     PARM 1         NUMBL            # Pre-Exit Programs
C                     PARM           QSZBM            Folder List
C                     PARM 0         NUMBM            # Folder List
C                     PARM           TEXTD            Text Description
C                     PARM '*USE'    PUBAUT           Public Authority
C                     PARM           QUSBN            Error Code
C* Check for errors returned in the error code parameter.
C                     EXSR ERRCOD
C                     ENDSR
C*
C*****************************************************************
C*****************************************************************
C*
C* Subroutine: COBJD
C*
C* Descriptive Name: Change object descriptions for the
C* ABC Product.
C*
C* Description: This subroutine changes the object
C*              descriptions for all objects that make up the
C*              ABC Product.  Currently, 15 objects exist.  They
C*              are listed at the end of this program.
C*
C*****************************************************************
C*****************************************************************
C*
C           COBJD     BEGSR
C*
C* Need to associate all objects with the ABC Product
C           1         DO   15        I       30
C                     MOVE OBJ,I     OBJDS
C           NAME      CAT  'ABC'     QOBJNM 20
C                     MOVELLP        LP5
C                     MOVELPID       PID13
C                     MOVELLID       LID12
C                     MOVELTYP       TYPE   10
C                     MOVEL'QLICOBJD'API
C                     CALL 'QLICOBJD'
C                     PARM           RTNLIB 10        Returned Lib. Name
C                     PARM           QOBJNM           Qual. Object Name
C                     PARM           TYPE             Object Type
C                     PARM           COBJI            Chg'd Object Info.
C                     PARM           QUSBN            Error Code
C* Check for any errors returned in the error code parameter.
C                     EXSR ERRCOD
C                     ENDDO
C                     ENDSR
C*
C*****************************************************************
C*****************************************************************
C*
C* Subroutine: PKGPO
C*
C* Descriptive Name: Package software ABC Product.
C*
C* Description: This subroutine packages the ABC Product.
```

```
     C*              It makes sure that all objects exist that are
     C*              associated with the product.
     C*
     C*****************************************************************
     C*****************************************************************
     C*
     C           PKGPO     BEGSR
     C*
     C* Setup for packing the ABC Product.
     C* Fill Product Option Information Parameter
     C                     MOVEL'0000'    QSZBRB           Product Option
     C                     MOVEL'0ABCABC' QSZBRC           Product ID
     C                     MOVEL'V3R1M0'  QSZBRD           Release Level
     C                     MOVEL'*ALL    'QSZBRF           Load ID
     C                     MOVEL*BLANKS   QSZBRG           Reserved
     C*
     C* Package the ABC Product.
     C*
     C*
     C                     MOVEL'QSZPKGPO'API
     C                     CALL 'QSZPKGPO'
     C                     PARM           QSZBR            Prod. Option Info.
     C                     PARM '*YES'    REPKG   4        Repackage
     C                     PARM '*NO  '   ALWCHG  5        Allow Object Change
     C                     PARM           QUSBN            Error Code
     C* Check for any errors returned in the error code parameter.
     C                     EXSR ERRCOD
     C                     ENDSR
     C*
     C*****************************************************************
     C*****************************************************************
     C*
     C* Subroutine: ERRCOD
     C*
     C* Descriptive Name: Process API errors.
     C*
     C* Description: This subroutine prints a line to a spooled
     C*              file if any errors are returned in the error code
     C*              parameter.
     C*
     C*****************************************************************
     C*****************************************************************
     C*
     C           ERRCOD    BEGSR
     C           QUSBNC    IFNE 0
     C*
     C* Process errors returned from the API.
     C*
     C                     EXCPTBADNWS
     C                     END
     C                     ENDSR
     OQPRINT  E  106           BADNWS
     O                                    'Failed in API '
     O                         API
     O                                    'with error '
     O                         QUSBND
     O* The information below is for array OBJ.
     O*111 represents the object name.
     O*2222222222 represents the object type.
     O*3333 represents the product option ID.
     O*4444 represents the product option load ID.
     O*5555555555555 represents the licensed program.
     O*11122222222223333444455555555555555
**
ABCPGMMRM1*PGM      000050010ABCABCV3R1M0
ABCPGMMRM2*PGM      000050010ABCABCV3R1M0
ABCPGMMRI1*PGM      000029240ABCABCV3R1M0
```

```
ABCPGMMRI2*PGM      000029240ABCABCV3R1M0
ABCPGM    *PGM      000050010ABCABCV3R1M0
QCLSRC    *FILE     000029240ABCABCV3R1M0
ABCDSPF   *FILE     000029240ABCABCV3R1M0
ABCPF     *FILE     000029240ABCABCV3R1M0
ABCMSG    *MSGF     000029240ABCABCV3R1M0
ABC       *CMD      000029240ABCABCV3R1M0
ABCPNLGRP *PNLGRP   000029240ABCABCV3R1M0
ABC0050   *PRDDFN   000050010ABCABCV3R1M0
ABC0050   *PRDLOD   000050010ABCABCV3R1M0
ABC0029   *PRDLOD   000029240ABCABCV3R1M0
ABC       *LIB      000050010ABCABCV3R1M0
```

Before you can build PTFs for the product, you need to save the product and install the product by using the Save Licensed Program (SAVLICPGM) and Restore Licensed Program (RSTLICPGM) commands.

After the product is built, you can perform the following tasks:

- Build PTFs for the product by using the following APIs:
  - Create Program Temporary Fix (QPZCRTFX)
  - Retrieve Program Temporary Fix Information (QPZRTVFX)
  - Program Temporary Fix Exit Program
- Use save, restore, or delete license program (SAVLICPGM, RSTLICPGM, DLTLICPGM) commands on it.
- Retrieve information about the product by using the Retrieve Product Information (QSZRTVPR) API.
- Check the product to verify the existence of libraries, folders, and objects that are part of the specified product (Check Product Option (CHKPRDOPT) command).

Related reference:

"Example in ILE C: Packaging a product"
This ILE C program creates objects 12 through 14 and packages your product.

"Example in ILE COBOL: Packaging a product" on page 253
This ILE COBOL program creates objects 12 through 14 and packages your product.

"Example in ILE RPG: Packaging a product" on page 260
This ILE RPG program creates objects 12 through 14 and packages your product.

Example in ILE C: Packaging a product:

This ILE C program creates objects 12 through 14 and packages your product.

Objects 12 through 14 are listed in Table 19 on page 238.

```
/*********************************************************************/
/* Program Name:          SFTWPRDEX                                  */
/*                                                                   */
/* Program Language:      ILE C                                      */
/*                                                                   */
/* Description:           This example shows you the steps necessary*/
/*                        to package your product like IBM's.        */
/*                                                                   */
/* Header Files Included: <stdlib.h>                                 */
/*                        <signal.h>                                 */
/*                        <string.h>                                 */
/*                        <stdio.h>                                  */
/*                        <qszcrtpd.h>                               */
/*                        <qszcrtpl.h>                               */
/*                        <qszpkgpo.h>                               */
/*                        <qlicobjd.h>                               */
/*                        <qusec.h>                                  */
/*                        <qliept.h>                                 */
/*                                                                   */
```

```
/* APIs Used:             QSZCRTPD - Create Product Definition      */
/*                        QSZCRTPL - Create Product Load            */
/*                        QSZPKGPO - Package Product Option         */
/*                        QLICOBJD - Change Object Description      */
/******************************************************************/
 #include <stdlib.h>
 #include <signal.h>
 #include <string.h>
 #include <stdio.h>
 #include <qszcrtpd.h>
 #include <qszcrtpl.h>
 #include <qszpkgpo.h>
 #include <qlicobjd.h>
 #include <qusec.h>
 #include <qliept.h>

/******************************************************************/
/* Function:     Create_Prod_Def_Obj                              */
/* Description:  Create the product definition ABC0050 for product  */
/*               ABC.                                             */
/******************************************************************/
void Create_Prod_Def_Obj()
{
  Qsz_Prd_Inf_t prod_info;           /* Product information         */
  Qsz_Prd_Opt_t prod_opt_list;       /* Product option list         */
  Qsz_Lng_Lod_t prod_lang_load;      /* Product language load list  */
  Qus_EC_t error_code;               /* Error code parameter        */
  char text_desc[50];                /* Text description            */

  /****************************************************************/
  /* Fill in the product information.                             */
  /****************************************************************/
  memset(&prod_info,' ',sizeof(prod_info));
  memcpy(prod_info.PID,"0ABCABC",7);
  memcpy(prod_info.Rls_Lvl,"V3R1M0",6);
  memcpy(prod_info.Msg_File,"ABCMSG    ",10);
  memcpy(prod_info.Fst_Cpyrt,"*CURRENT  ",10);
  memcpy(prod_info.Cur_Cpyrt,"*CURRENT  ",10);
  memcpy(prod_info.Rls_Date,"941201",6);
  memcpy(prod_info.Alw_Mult_Rls,"*NO ",4);
  memcpy(prod_info.Reg_ID_Type,"*PHONE    ",10);
  memcpy(prod_info.Reg_ID_Val,"5072530927    ",14);
  /****************************************************************/
  /* Fill in the product option list.                             */
  /****************************************************************/
  memset(&prod_opt_list,' ',sizeof(prod_opt_list));
  memcpy(prod_opt_list.Opt,"0000",4);
  memcpy(prod_opt_list.Msg_ID,"ABC0001",7);
  memcpy(prod_opt_list.Alw_Dyn_Nam,"*NODYNNAM ",10);
  memcpy(prod_opt_list.Cod_Lod,"5001",4);
  /****************************************************************/
  /* Fill in the product language load list.                      */
  /****************************************************************/
  memset(&prod_lang_load,' ',sizeof(prod_lang_load));
  memcpy(prod_lang_load.Lng_Lod,"2924    ",8);
  memcpy(prod_lang_load.Opt,"0000",4);

  memset(text_desc,' ',50);
  memcpy(text_desc,"Product ABC",11);

  /****************************************************************/
  /* Initialize the error code to have the API send errors through  */
  /* the error code parameter.                                    */
  /****************************************************************/
  error_code.Bytes_Provided=sizeof(error_code);
  QSZCRTPD("ABC0050   ABC       ",   /* Product definition name     */
           &prod_info,               /* Product definition info     */
```

```
          &prod_opt_list,           /* Product option list         */
          1,                        /* Number of options           */
          &prod_lang_load,          /* Language load list          */
          1,                        /* Number languages            */
          text_desc,                /* Text description            */
          "*USE      ",             /* Public authority            */
          &error_code);             /* Error code                  */

  if (error_code.Bytes_Available > 0)
  {
     printf("Failed in QSZCRTPD API with error: %.7s",
             error_code.Exception_Id);
     exit(1);
  }
}

/*****************************************************************/
/* Function:     Create_Prod_Load_Obj                            */
/* Description:  Create the product loads ABC0050 (MRM object) and  */
/*               ABC0029 (MRI object) for product ABC.           */
/*****************************************************************/
void Create_Prod_Load_Obj()
{
  Qsz_Lod_Inf_t prod_load_info;      /* Product load information    */
  Qsz_Lib_Inf_t prin_lib_info;       /* Principal library info      */
  Qsz_Add_Lib_t add_libs;            /* Additional library list     */
  Qsz_Pre_Ext_t preop_expgm;         /* Preoperational exit program */
  Qsz_Flr_Lst_t folder_list;         /* Folder list                 */
  Qus_EC_t error_code;               /* Error code parameter        */
  char text_desc[50];                /* Text description            */


  /*****************************************************************/
  /* Fill in the product load information.                         */
  /*****************************************************************/
  memset(&prod_load_info,' ',sizeof(prod_load_info));
  memcpy(prod_load_info.PID,"0ABCABC",7);
  memcpy(prod_load_info.Rls_Lvl,"V3R1M0",6);
  memcpy(prod_load_info.Opt,"0000",4);
  memcpy(prod_load_info.Lod_Type,"*CODE     ",10);
  memcpy(prod_load_info.Lod_ID,"*CODEDFT",8);
  memcpy(prod_load_info.Reg_ID_Type,"*PRDDFN   ",10);
  memcpy(prod_load_info.Min_Tgt_Rls,"*CURRENT  ",10);

  /*****************************************************************/
  /* Fill in the principal library information.  There are no      */
  /* additional libraries.                                         */
  /*****************************************************************/
  memcpy(prin_lib_info.Dev_Lib,"ABC       ",10);
  memcpy(prin_lib_info.Prim_Lib,"ABC       ",10);
  memcpy(prin_lib_info.Post_Exit_Pgm,"ABCPGMMRM2",10);

  memset(&add_libs,' ',sizeof(add_libs));

  /*****************************************************************/
  /* Fill in the preoperational exit program.                      */
  /*****************************************************************/
  memcpy(preop_expgm.Pre_Ext_Pgm,"ABCPGMMRM1",10);
  memcpy(preop_expgm.Dev_Lib,"ABC       ",10);

  /*****************************************************************/
  /* There are no folders.                                         */
  /*****************************************************************/
  memset(&folder_list,' ',sizeof(folder_list));

  memset(text_desc,' ',50);
  memcpy(text_desc,"Product ABC",11);
```

```
  /*****************************************************************/
  /* Initialize the error code to have the API send errors through  */
  /* the error code parameter.                                      */
  /*****************************************************************/
  error_code.Bytes_Provided=sizeof(error_code);
  QSZCRTPL("ABC0050   ",             /* Product load name           */
           &prod_load_info,          /* Product load information    */
           "          ",             /* Secondary language lib name */
           &prin_lib_info,           /* Principal library           */
           &add_libs,                /* Additional libraries        */
           0,                        /* Number of additional libs   */
           &preop_expgm,             /* Preoperational exit program */
           1,                        /* Number of preop exit pgms   */
           &folder_list,             /* Folder list                 */
           0,                        /* Number of folders           */
           text_desc,                /* Text description            */
           "*USE      ",             /* Public authority            */
           &error_code);             /* Error code                  */

  if (error_code.Bytes_Available > 0)
  {
     printf("Failed in QSZCRTPL API with error: %.7s",
             error_code.Exception_Id);
     exit(1);
  }

  /*****************************************************************/
  /* Fill in the product load information.                         */
  /*****************************************************************/
  memcpy(prod_load_info.Lod_Type,"*LNG      ",10);
  memcpy(prod_load_info.Lod_ID,"2924    ",8);

  /*****************************************************************/
  /* Fill in the principal library information.  There are no      */
  /* additional libraries.                                         */
  /*****************************************************************/
  memcpy(prin_lib_info.Post_Exit_Pgm,"ABCPGMMRI2",10);

  /*****************************************************************/
  /* Fill in the preoperational exit program.                      */
  /*****************************************************************/
  memcpy(preop_expgm.Pre_Ext_Pgm,"ABCPGMMRI1",10);

  QSZCRTPL("ABC0029   ",             /* Product load name           */
           &prod_load_info,          /* Product load information    */
           "ABC2924   ",             /* Secondary language lib name */
           &prin_lib_info,           /* Principal library           */
           &add_libs,                /* Additional libraries        */
           0,                        /* Number of additional libs   */
           &preop_expgm,             /* Preoperational exit program */
           1,                        /* Number of preop exit pgms   */
           &folder_list,             /* Folder list                 */
           0,                        /* Number of folders           */
           text_desc,                /* Text description            */
           "*USE      ",             /* Public authority            */
           &error_code);             /* Error code                  */

  if (error_code.Bytes_Available > 0)
  {
     printf("Failed in QSZCRTPL API with error: %.7s",
             error_code.Exception_Id);
     exit(1);
  }

}
```

```
/*********************************************************************/
/* Function:     Change_Obj_Descr                                    */
/* Description:  Change object descriptions for all objects          */
/*               that make up Product ABC.  Currently there are 15   */
/*               objects.                                            */
/*********************************************************************/
void Change_Obj_Descr()
{
  typedef struct {
    char obj_name_lib[21];
    char obj_type[11];
    char prd_opt_id[5];
    char prd_opt_ld[5];
    char lp_id[4];
  } obj_info_t;

  typedef struct {
    int numkey;
    Qus_Vlen_Rec_3_t PID_rec;
    char PID[4];
    Qus_Vlen_Rec_3_t LID_rec;
    char LID[4];
    Qus_Vlen_Rec_3_t LP_rec;
    char LP[13];
  } change_obj_info_t;

  int i;
  obj_info_t obj_info[15] = {"ABCPGMMRM1ABC       ","*PGM      ",
                             "0000","5001","0ABCABCV3R1M0",
                             "ABCPGMMRM2ABC       ","*PGM      ",
                             "0000","5001","0ABCABCV3R1M0",
                             "ABCPGMMRI1ABC       ","*PGM      ",
                             "0000","2924","0ABCABCV3R1M0",
                             "ABCPGMMRI2ABC       ","*PGM      ",
                             "0000","2924","0ABCABCV3R1M0",
                             "ABCPGM    ABC       ","*PGM      ",
                             "0000","5001","0ABCABCV3R1M0",
                             "QCLSRC    ABC       ","*FILE     ",
                             "0000","2924","0ABCABCV3R1M0",
                             "ABCDSPF   ABC       ","*FILE     ",
                             "0000","2924","0ABCABCV3R1M0",
                             "ABCPF     ABC       ","*FILE     ",
                             "0000","2924","0ABCABCV3R1M0",
                             "ABCMSG    ABC       ","*MSGF     ",
                             "0000","2924","0ABCABCV3R1M0",
                             "ABC       ABC       ","*CMD      ",
                             "0000","2924","0ABCABCV3R1M0",
                             "ABCPNLGRP ABC       ","*PNLGRP   ",
                             "0000","2924","0ABCABCV3R1M0",
                             "ABC0050   ABC       ","*PRDDFN   ",
                             "0000","5001","0ABCABCV3R1M0",
                             "ABC0050   ABC       ","*PRDLOD   ",
                             "0000","5001","0ABCABCV3R1M0",
                             "ABC0029   ABC       ","*PRDLOD   ",
                             "0000","2924","0ABCABCV3R1M0",
                             "ABC       ABC       ","*LIB      ",
                             "0000","5001","0ABCABCV3R1M0"};
  change_obj_info_t cobji;           /* Change object information   */
  Qus_EC_t error_code;               /* Error code parameter        */
  char rtn_lib[10];                  /* Return library              */

  /*******************************************************************/
  /* Fill in the changed object information.                         */
  /*******************************************************************/
  cobji.numkey=3;
  cobji.PID_rec.Key=13;
  cobji.PID_rec.Length_Vlen_Record=4;
```

```
  cobji.LID_rec.Key=12;
  cobji.LID_rec.Length_Vlen_Record=4;
  cobji.LP_rec.Key=5;
  cobji.LP_rec.Length_Vlen_Record=13;

  /*****************************************************************/
  /* Initialize the error code to have the API send errors through  */
  /* the error code parameter.                                     */
  /*****************************************************************/
  error_code.Bytes_Provided=sizeof(error_code);

  for (i=0; i<15; i++)
  {
    memcpy(cobji.PID,obj_info[i].prd_opt_id,4);
    memcpy(cobji.LID,obj_info[i].prd_opt_ld,4);
    memcpy(cobji.LP,obj_info[i].lp_id,13);

    QLICOBJD(rtn_lib,                 /* Return library            */
             obj_info[i].obj_name_lib, /* Object name              */
             obj_info[i].obj_type,     /* Object type              */
             &cobji,                   /* Changed object information*/
             &error_code);             /* Error code               */

    if (error_code.Bytes_Available > 0)
    {
       printf("Failed in QLICOBJD API with error: %.7s",
               error_code.Exception_Id);
       exit(1);
    }
  }
}

/*********************************************************************/
/* Function:     Package_Prod                                        */
/* Description:  Package Product ABC so that all the SAVLICPGM,     */
/*               RSTLICPGM and DLTLICPGM commands work with the     */
/*               product.                                            */
/*********************************************************************/
void Package_Prod()
{
  Qsz_Prd_Opt_Inf_t prod_opt_info;   /* Product option information  */
  Qus_EC_t error_code;               /* Error code parameter        */



  /*****************************************************************/
  /* Fill in the product option information.                       */
  /*****************************************************************/
  memset(&prod_opt_info,' ',sizeof(prod_opt_info));
  memcpy(prod_opt_info.Opt,"0000",4);
  memcpy(prod_opt_info.PID,"0ABCABC",7);
  memcpy(prod_opt_info.Rls_Lvl,"V3R1M0",6);
  memcpy(prod_opt_info.Lod_ID,"*ALL    ",8);

  /*****************************************************************/
  /* Initialize the error code to have the API send errors through  */
  /* the error code parameter.                                     */
  /*****************************************************************/
  error_code.Bytes_Provided=sizeof(error_code);
  QSZPKGPO(&prod_opt_info,           /* Product option information  */
           "*YES",                   /* Repackage                  */
           "*NO  ",                  /* Allow object change        */
           &error_code);             /* Error code                 */

  if (error_code.Bytes_Available > 0)
  {
     printf("Failed in QSZPKGPO API with error: %.7s",
```

```
            error_code.Exception_Id);
    exit(1);
  }
}

/*****************************************************************/
/* Start of main procedure                                       */
/*****************************************************************/

void main()
{

  /*****************************************************************/
  /* Create Product Definition Object                              */
  /*****************************************************************/
  Create_Prod_Def_Obj();

  /*****************************************************************/
  /* Create Product Load Objects                                   */
  /*****************************************************************/
  Create_Prod_Load_Obj();

  /*****************************************************************/
  /* Change Object Description                                     */
  /*****************************************************************/
  Change_Obj_Descr();

  /*****************************************************************/
  /* Package Product ABC                                           */
  /*****************************************************************/
  Package_Prod();

}
```

Related reference:

"Example in OPM RPG: Packaging a product" on page 240
This OPM RPG program creates objects 12 through 14 and packages your product.

Example in ILE COBOL: Packaging a product:

This ILE COBOL program creates objects 12 through 14 and packages your product.

Objects 12 through 14 are listed in Table 19 on page 238.

The following program also works for OPM COBOL.

```
       IDENTIFICATION DIVISION.
      ******************************************************************
      ******************************************************************
      *
      *Program Name: SFTWPRDEX
      *
      *Language: COBOL
      *
      *Descriptive Name: Software Product Example
      *
      *Description: This example shows you the steps necessary to
      *             package your product like IBM products.
      *
      *Header Files Included: QUSEC    - Error Code Parameter
      *                       QSZCRTPD - Create Product Definition API
      *                       QSZCRTPL - Create Product Load API
      *                       QSZPKGPO - Package Product Option API
      *
      ******************************************************************
      ******************************************************************
```

```
*
 PROGRAM-ID. SFTWPRDEX.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT LISTING ASSIGN TO PRINTER-QPRINT
                    ORGANIZATION IS SEQUENTIAL.
 DATA DIVISION.
 FILE SECTION.
 FD  LISTING RECORD CONTAINS 132 CHARACTERS
             LABEL RECORDS ARE STANDARD
             DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
*
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Create Product Definition API Include
*
 COPY QSZCRTPD OF QSYSINC-QLBLSRC.
*
* Create Product Load API Include
*
 COPY QSZCRTPL OF QSYSINC-QLBLSRC.
*
* Package Product Option API Include
*
 COPY QSZPKGPO OF QSYSINC-QLBLSRC.
*
* Error message text
*
 01  BAD-NEWS.
     05  TEXT1        PIC  X(14) VALUE "Failed in API ".
     05  API-NAME     PIC  X(10).
     05  TEXT2        PIC  X(11) VALUE "with error ".
     05  EXCEPTION-ID PIC X(07).
*
* Compile Time Array
*
 01  OBJ-INFO.
     05 ELEMENT-01 PIC X(41)
              VALUE "ABCPGMMRM1*PGM      000050010ABCABCV3R1M0".
     05 ELEMENT-02 PIC X(41)
              VALUE "ABCPGMMRM2*PGM      000050010ABCABCV3R1M0".
     05 ELEMENT-03 PIC X(41)
              VALUE "ABCPGMMRI1*PGM      000029240ABCABCV3R1M0".
     05 ELEMENT-04 PIC X(41)
              VALUE "ABCPGMMRI2*PGM      000029240ABCABCV3R1M0".
     05 ELEMENT-05 PIC X(41)
              VALUE "ABCPGM    *PGM      000050010ABCABCV3R1M0".
     05 ELEMENT-06 PIC X(41)
              VALUE "QCLSRC    *FILE     000029240ABCABCV3R1M0".
     05 ELEMENT-07 PIC X(41)
              VALUE "ABCDSPF   *FILE     000029240ABCABCV3R1M0".
     05 ELEMENT-08 PIC X(41)
              VALUE "ABCPF     *FILE     000029240ABCABCV3R1M0".
     05 ELEMENT-09 PIC X(41)
              VALUE "ABCMSG    *MSGF     000029240ABCABCV3R1M0".
     05 ELEMENT-10 PIC X(41)
```

```
              VALUE "ABC       *CMD      000029240ABCABCV3R1M0".
     05 ELEMENT-11 PIC X(41)
              VALUE "ABCPNLGRP *PNLGRP   000029240ABCABCV3R1M0".
     05 ELEMENT-12 PIC X(41)
              VALUE "ABC0050   *PRDDFN   000050010ABCABCV3R1M0".
     05 ELEMENT-13 PIC X(41)
              VALUE "ABC0050   *PRDLOD   000050010ABCABCV3R1M0".
     05 ELEMENT-14 PIC X(41)
              VALUE "ABC0029   *PRDLOD   000029240ABCABCV3R1M0".
     05 ELEMENT-15 PIC X(41)
              VALUE "ABC       *LIB      000050010ABCABCV3R1M0".
*
 01  OBJECT-TABLE REDEFINES OBJ-INFO.
     05 OBJ-INFO-I OCCURS 15 TIMES.
        10 OBJ-NAME      PIC  X(10).
        10 OBJ-TYPE      PIC  X(10).
        10 PRD-OPT-ID    PIC  X(04).
        10 PRD-OPT-LD    PIC  X(04).
        10 LP-ID         PIC  X(13).
*
* Change Object Information parameter
*
 01  COBJI.
     05  NUMKEY          PIC S9(09) VALUE 3 BINARY.
     05  KEY13           PIC S9(09) VALUE 13 BINARY.
     05  LEN13           PIC S9(09) VALUE 4 BINARY.
     05  PID13           PIC  X(04).
     05  KEY12           PIC S9(09) VALUE 12 BINARY.
     05  LEN12           PIC S9(09) VALUE 4 BINARY.
     05  LID12           PIC  X(04).
     05  KEY5            PIC S9(09) VALUE 5 BINARY.
     05  LEN5            PIC S9(09) VALUE 13 BINARY.
     05  LP5             PIC  X(13).
*
* Miscellaneous data
*
 01  MISC.
     05  FIRST-ERR       PIC  X(01) VALUE "0".
     05  PROD-ID         PIC  X(07) VALUE "0ABCABC".
     05  PROD-NAME       PIC  X(20) VALUE "ABC0050   ABC".
     05  RLS-LVL         PIC  X(06) VALUE "V3R1M0".
     05  NBR-OPTS        PIC S9(09) VALUE 1 BINARY.
     05  NBR-LANGS       PIC S9(09) VALUE 1 BINARY.
     05  TEXT-DESC       PIC  X(50) VALUE "ABC Product".
     05  PUB-AUT         PIC  X(10) VALUE "*USE".
     05  NBR-ADD-LB      PIC S9(09) VALUE 0 BINARY.
     05  NBR-PE          PIC S9(09) VALUE 1 BINARY.
     05  NBR-FLDRS       PIC S9(09) VALUE 0 BINARY.
     05  OBJNAM          PIC  X(20).
     05  PROD-ID-NM      PIC  X(10).
     05  SEC-LANG        PIC  X(10).
     05  I               PIC S9(09) BINARY.
     05  RTN-LIB         PIC  X(10).
     05  OBJ-TYPE-2      PIC  X(10).
     05  REPKG           PIC  X(04) VALUE "*YES".
     05  ALWCHG          PIC  X(05) VALUE "*NO".
*
* Beginning of Mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
```

```
* API for the parameter.
*
     MOVE LENGTH OF QUS-EC TO BYTES-PROVIDED OF QUS-EC.
*
* Create Product Definition Object - ABC0050
*
     PERFORM PRDDFN.  (1)
*
* Create Product Load Objects - ABC0050 (MRM) and ABC0029 (MRI)
*
     PERFORM PRDLOD.  (2)
*
* Change Object Description for all objects associated with
* ABC Product.
*
     PERFORM COBJD.  (3)
*
* Package the ABC Product so that all the SAVLICPGM, RSTLIBPGM,
* and DLTLICPGM commands work with the product.
*
     PERFORM PKGPO.  (4)
*
* All done, product is ready to ship.
*
     STOP RUN.
*
* End of MAINLINE
*
*****************************************************************
*****************************************************************
*
* Subroutine: PRDDFN
*
* Descriptive Name: Create product definitions.
*
* Description: This subroutine will create the product definition
*              ABC0050 for the ABC Product.
*
*****************************************************************
*****************************************************************
*
 PRDDFN.
*
* Setup for Product Definition
* Fill Product Definition Information Parameter
*
     MOVE PROD-ID OF MISC TO PID OF QSZ-PRD-INF.
     MOVE RLS-LVL OF MISC TO RLS-LVL OF QSZ-PRD-INF.
     MOVE "ABCMSG" TO MSG-FILE OF QSZ-PRD-INF.
     MOVE "*CURRENT" TO FST-CPYRT OF QSZ-PRD-INF.
     MOVE "*CURRENT" TO CUR-CPYRT OF QSZ-PRD-INF.
     MOVE "941201" TO RLS-DATE OF QSZ-PRD-INF.
     MOVE "*NO" TO ALW-MULT-RLS OF QSZ-PRD-INF.
     MOVE "*PHONE" TO REG-ID-TYPE OF QSZ-PRD-INF.
     MOVE "5072535010" TO REG-ID-VAL OF QSZ-PRD-INF.
*
* Fill Product Load Parameter
*
     MOVE "0000" TO OPT OF QSZ-PRD-OPT.
     MOVE "ABC0001" TO MSG-ID OF QSZ-PRD-OPT.
     MOVE "*NODYNNAM" TO ALW-DYN-NAM OF QSZ-PRD-OPT.
     MOVE "5001" TO COD-LOD OF QSZ-PRD-OPT.
     MOVE SPACES TO RESERVED OF QSZ-PRD-OPT.
*
* Fill Language Load List Parameter
*
     MOVE "2924" TO LNG-LOD OF QSZ-LNG-LOD.
```

```
     MOVE "0000" TO OPT OF QSZ-LNG-LOD.
     MOVE SPACES TO RESERVED OF QSZ-LNG-LOD.
*
* Create the Product Definition for the ABC Product
*
     MOVE 1 TO NBR-OPTS.
     MOVE 1 TO NBR-LANGS.
     CALL "QSZCRTPD" USING PROD-NAME, QSZ-PRD-INF, QSZ-PRD-OPT,
                           NBR-OPTS, QSZ-LNG-LOD, NBR-LANGS,
                           TEXT-DESC, PUB-AUT, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                        MOVE "QSZCRTPD" TO API-NAME,
                        PERFORM ERRCOD.
*
*****************************************************************
*****************************************************************
*
* Subroutine: PRDLOD
*
* Descriptive Name: Create product loads.
*
* Description: This subroutine will create the product loads,
*              ABC0050 and ABC0029, for the ABC Product.
*
*****************************************************************
*****************************************************************
*
 PRDLOD.
*
* Setup for Product Load for MRM Objects
* Fill Product Load Information Parameter
*
     MOVE PROD-ID OF MISC TO PID OF QSZ-LOD-INF.
     MOVE RLS-LVL OF MISC TO RLS-LVL OF QSZ-LOD-INF.
     MOVE "0000" TO OPT OF QSZ-LOD-INF.
     MOVE "*CODE" TO LOD-TYPE OF QSZ-LOD-INF.
     MOVE "*CODEDFT" TO LOD-ID OF QSZ-LOD-INF.
     MOVE "*PRDDFN" TO REG-ID-TYPE OF QSZ-LOD-INF.
     MOVE SPACES TO REG-ID-VAL OF QSZ-LOD-INF.
     MOVE "*CURRENT" TO MIN-TGT-RLS OF QSZ-LOD-INF.
     MOVE SPACES TO RESERVED OF QSZ-LOD-INF.
*
* Fill Principal Library Information Parameter
*
     MOVE "ABC" TO DEV-LIB OF QSZ-LIB-INF.
     MOVE "ABC" TO PRIM-LIB OF QSZ-LIB-INF.
     MOVE "ABCPGMMRM2" TO POST-EXIT-PGM OF QSZ-LIB-INF.
*
* Fill Preoperation Exit Programs Parameter
*
     MOVE "ABCPGMMRM1" TO PRE-EXT-PGM OF QSZ-PRE-EXT.
     MOVE "ABC" TO DEV-LIB OF QSZ-PRE-EXT.
*
* Fill Additional Library List Parameter
*       None
*
* Fill Folder List Parameter
*       None
*
* Let's create the product load for the ABC Product - MRM Objects
*
```

```
     MOVE "ABC0050" TO PROD-ID-NM.
     MOVE SPACES TO SEC-LANG.
*
     CALL "QSZCRTPL" USING PROD-ID-NM, QSZ-LOD-INF, SEC-LANG,
                           QSZ-LIB-INF, QSZ-ADD-LIB,
                           NBR-ADD-LB, QSZ-PRE-EXT, NBR-PE,
                           QSZ-FLR-LST, NBR-FLDRS, TEXT-DESC,
                           PUB-AUT, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                        MOVE "QSZCRTPL" TO API-NAME,
                        PERFORM ERRCOD.
*
* Setup for Product Load for MRI Objects
* Fill Product Load Information Parameter
*
     MOVE "*LNG" TO LOD-TYPE OF QSZ-LOD-INF.
     MOVE "2924" TO LOD-ID OF QSZ-LOD-INF.
*
* Fill Principal Library Information Parameter
*
     MOVE "ABCPGMMRI2" TO POST-EXIT-PGM OF QSZ-LIB-INF.
*
* Fill Preoperation Exit Programs Parameter
*
     MOVE "ABCPGMMRI1" TO PRE-EXT-PGM OF QSZ-PRE-EXT.
*
* Fill Additional Library List Parameter
*       None
*
* Fill Folder List Parameter
*       None
*
* Let's create the product load for the ABC Product - MRI Objects
*
     MOVE "ABC0029" TO PROD-ID-NM.
     MOVE "ABC2924" TO SEC-LANG.
*
     CALL "QSZCRTPL" USING PROD-ID-NM, QSZ-LOD-INF, SEC-LANG,
                           QSZ-LIB-INF, QSZ-ADD-LIB,
                           NBR-ADD-LB, QSZ-PRE-EXT, NBR-PE,
                           QSZ-FLR-LST, NBR-FLDRS, TEXT-DESC,
                           PUB-AUT, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                        MOVE "QSZCRTPL" TO API-NAME,
                        PERFORM ERRCOD.
*
****************************************************************
****************************************************************
*
* Subroutine: COBJD
*
* Descriptive Name: Change object descriptions for ABC Product.
*
* Description: This subroutine will change the object
*              descriptions for all objects that make up the
```

```
*              ABC Product.  Currently that is 15 objects.  They
*              are listed at the end of this program.
*
****************************************************************
****************************************************************
*
 COBJD.
*
* Need to associate all objects with the ABC Product
*
     PERFORM CHG-OBJD VARYING I FROM 1 BY 1 UNTIL I > 15.
*
 CHG-OBJD.
     STRING OBJ-NAME(I), "ABC" DELIMITED BY SIZE INTO OBJNAM.
     MOVE LP-ID(I) TO LP5.
     MOVE PRD-OPT-ID(I) TO PID13.
     MOVE PRD-OPT-LD(I) TO LID12.
     MOVE OBJ-TYPE(I) TO OBJ-TYPE-2.
*
     CALL "QLICOBJD" USING RTN-LIB, OBJNAM, OBJ-TYPE-2,
                           COBJI, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                        MOVE "QLICOBJD" TO API-NAME,
                        PERFORM ERRCOD.
****************************************************************
****************************************************************
*
* Subroutine: PKGPO
*
* Descriptive Name: Package software ABC Product.
*
* Description: This subroutine will package the ABC Product.
*              It makes sure that all objects exist that are
*              associated with the product.
*
****************************************************************
****************************************************************
*
 PKGPO.
*
* Setup for packing the ABC Product.
* Fill Product Option Information Parameter
*
     MOVE "0000" TO OPT OF QSZ-PRD-OPT-INF.
     MOVE PROD-ID OF MISC TO PID OF QSZ-PRD-OPT-INF.
     MOVE RLS-LVL OF MISC TO RLS-LVL OF QSZ-PRD-OPT-INF.
     MOVE "*ALL" TO LOD-ID OF QSZ-PRD-OPT-INF.
     MOVE SPACES TO RESERVED OF QSZ-PRD-OPT-INF.
*
* Let's package the ABC Product.
*
     CALL "QSZPKGPO" USING QSZ-PRD-OPT-INF, REPKG,
                           ALWCHG, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                        MOVE "QSZPKGPO" TO API-NAME,
```

```
                        PERFORM ERRCOD.
*
****************************************************************
****************************************************************
*
* Subroutine: ERRCOD
*
* Descriptive Name: Process API errors.
*
* Description: This subroutine will print a line to a spooled
*              file if any errors are returned in the error code
*              parameter.
*
****************************************************************
****************************************************************
*
 ERRCOD.
*
* Process errors returned from the API.
*
* If first error found, then open QPRINT *PRTF
*
     IF FIRST-ERR = "0"
                     OPEN OUTPUT LISTING,
                     MOVE "1" TO FIRST-ERR.
*
* Output the error and the API that received the error
*
     MOVE EXCEPTION-ID OF QUS-EC TO EXCEPTION-ID OF BAD-NEWS.
     WRITE LIST-LINE FROM BAD-NEWS.
```

Related reference:

"Example in OPM RPG: Packaging a product" on page 240
This OPM RPG program creates objects 12 through 14 and packages your product.

Example in ILE RPG: Packaging a product:

This ILE RPG program creates objects 12 through 14 and packages your product.

Objects 12 through 14 are listed in Table 19 on page 238.

```
F****************************************************************
F****************************************************************
F*
F*Program Name: SFTWPRDEX
F*
F*Language: ILE RPG
F*
F*Descriptive Name: Software Product Example
F*
F*Description: This example shows you the steps necessary to
F*             package your product like IBM products.
F*
F*Header Files Included: QUSEC    - Error Code Parameter
F*                       QSZCRTPD - Create Product Definition API
F*                       QSZCRTPL - Create Product Load API
F*                       QSZPKGPO - Package Product Option API
F*
F****************************************************************
F****************************************************************
F*
FQPRINT    O    F  132        PRINTER OFLIND(*INOF) USROPN
D*
D* Error Code parameter include.  As this sample program
D* uses /COPY to include the error code structure, only the first
D* 16 bytes of the error code structure are available.  If the
```

```
D* application program needs to access the variable length
D* exception data for the error, the developer should physically
D* copy the QSYSINC include and modify the copied include to
D* define additional storage for the exception data.
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D* Create Product Definition API Include
D*
D/COPY QSYSINC/QRPGLESRC,QSZCRTPD
D*
D* Create Product Load API Include
D*
D/COPY QSYSINC/QRPGLESRC,QSZCRTPL
D*
D* Package Product Option API Include
D*
D/COPY QSYSINC/QRPGLESRC,QSZPKGPO
D*
D* Compile Time Array
D*
DOBJ_INFO         S             41    DIM(15) CTDATA PERRCD(1)
D*
DOBJ_INFO_I       DS                  BASED(OBJ_PTR)
D OBJ_NAME                      10
D OBJ_TYPE                      10
D PRD_OPT_ID                     4
D PRD_OPT_LD                     4
D LP_ID                         13
D*
D* Change Object Information parameter
D*
DCOBJI            DS
D NUMKEY                         9B 0 INZ(3)
D KEY13                          9B 0 INZ(13)
D LEN13                          9B 0 INZ(4)
D PID13                          4
D KEY12                          9B 0 INZ(12)
D LEN12                          9B 0 INZ(4)
D LID12                          4
D KEY5                           9B 0 INZ(5)
D LEN5                           9B 0 INZ(13)
D LP5                           13
D*
D* Miscellaneous data
D*
DAPI_NAME         S             10
DFIRST_ERR        S              1    INZ('0')
DPROD_ID          S              7    INZ('0ABCABC')
DPROD_NAME        S             20    INZ('ABC0050   ABC       ')
DRLS_LVL          S              6    INZ('V3R1M0')
DNBR_OPTS         S              9B 0 INZ(1)
DNBR_LANGS        S              9B 0 INZ(1)
DTEXT_DESC        S             50    INZ('ABC Product')
DPUB_AUT          S             10    INZ('*USE')
DNBR_ADD_LB       S              9B 0 INZ(0)
DNBR_PE           S              9B 0 INZ(1)
DNBR_FLDRS        S              9B 0 INZ(0)
DOBJNAM           S             20
C*
C* Beginning of Mainline
C*
C* Initialize the error code parameter.  To signal exceptions to
C* this program by the API, you need to set the bytes provided
C* field of the error code to zero.  Because this program has
C* exceptions sent back through the error code parameter, it sets
C* the bytes provided field to the number of bytes it gives the
```

```
C* API for the parameter.
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Create Product Definition Object - ABC0050
C*
C                   EXSR      PRDDFN     (1)
C*
C* Create Product Load Objects - ABC0050 (MRM) and ABC0029 (MRI)
C*
C                   EXSR      PRDLOD     (2)
C*
C* Change Object Description for all objects associated with
C* the ABC Product.
C*
C                   EXSR      COBJD      (3)
C*
C* Package the ABC Product so that all the SAVLICPGM, RSTLIBPGM,
C* and DLTLICPGM commands work with the product.
C*
C                   EXSR      PKGPO      (4)
C*
C* All done, product is ready to ship.
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C*
C* End of MAINLINE
C*
C*
C*****************************************************************
C*****************************************************************
C*
C* Subroutine: PRDDFN
C*
C* Descriptive Name: Create product definitions.
C*
C* Description: This subroutine will create the product definition
C*              ABC0050 for the ABC product.
C*
C*****************************************************************
C*****************************************************************
C*
C     PRDDFN        BEGSR
C*
C* Setup for Product Definition
C* Fill Product Definition Information Parameter
C*
C                   EVAL      QSZPID = PROD_ID
C                   EVAL      QSZRL = RLS_LVL
C                   EVAL      QSZMFIL = 'ABCMSG'
C                   EVAL      QSZFC = '*CURRENT'
C                   EVAL      QSZCC = '*CURRENT'
C                   EVAL      QSZRD = '941201'
C                   EVAL      QSZAMR = '*NO'
C                   EVAL      QSZRIDT = '*PHONE'
C                   EVAL      QSZRIDV = '5072535010'
C*
C* Fill Product Load Parameter
C*
C                   EVAL      QSZOPT = '0000'
C                   EVAL      QSZMID = 'ABC0001'
C                   EVAL      QSZADN = '*NODYNNAM'
C                   EVAL      QSZCL = '5001'
C                   EVAL      QSZERVED00 = *BLANKS
C*
C* Fill Language Load List Parameter
```

```
C*
C                   EVAL      QSZLL00 = '2924'
C                   EVAL      QSZOPT00 = '0000'
C                   EVAL      QSZERVED01 = *BLANKS
C*
C* Create the Product Definition for the ABC Product
C*
C                   CALL      'QSZCRTPD'
C                   PARM                    PROD_NAME
C                   PARM                    QSZPI
C                   PARM                    QSZPO
C                   PARM      1             NBR_OPTS
C                   PARM                    QSZLL
C                   PARM      1             NBR_LANGS
C                   PARM                    TEXT_DESC
C                   PARM                    PUB_AUT
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   EVAL      API_NAME = 'QSZCRTPD'
C                   EXSR      ERRCOD
C                   ENDIF
C*
C                   ENDSR
C*
C*****************************************************************
C*****************************************************************
C*
C* Subroutine: PRDLOD
C*
C* Descriptive Name: Create product loads.
C*
C* Description: This subroutine will create the product loads,
C*              ABC0050 and ABC0029, for the ABC product.
C*
C*****************************************************************
C*****************************************************************
C*
C     PRDLOD        BEGSR
C*
C* Setup for Product Load for MRM Objects
C* Fill Product Load Information Parameter
C*
C                   EVAL      QSZPID00 = PROD_ID
C                   EVAL      QSZRL00 = RLS_LVL
C                   EVAL      QSZOPT01 = '0000'
C                   EVAL      QSZLT = '*CODE'
C                   EVAL      QSZLID = '*CODEDFT'
C                   EVAL      QSZRIDT00 = '*PRDDFN'
C                   EVAL      QSZRIDV00 = *BLANKS
C                   EVAL      QSZMTR = '*CURRENT'
C                   EVAL      QSZERVED02 = *BLANKS
C*
C* Fill Principal Library Information Parameter
C*
C                   EVAL      QSZDL = 'ABC'
C                   EVAL      QSZPL = 'ABC'
C                   EVAL      QSZPEP = 'ABCPGMMRM2'
C*
C* Fill Preoperation Exit Programs Parameter
C*
C                   EVAL      QSZPEP00 = 'ABCPGMMRM1'
```

```
C                   EVAL      QSZDL00 = 'ABC'
C*
C* Fill Additional Library List Parameter
C*       None
C*
C* Fill Folder List Parameter
C*       None
C*
C* Let's create the product load for the ABC Product - MRM Objects
C*
C                   CALL      'QSZCRTPL'
C                   PARM      'ABC0050'     PROD_ID_NM       10
C                   PARM                    QSZLI
C                   PARM      *BLANKS       SEC_LANG         10
C                   PARM                    QSZLI00
C                   PARM                    QSZAL
C                   PARM                    NBR_ADD_LB
C                   PARM                    QSZPE
C                   PARM                    NBR_PE
C                   PARM                    QSZFL
C                   PARM                    NBR_FLDRS
C                   PARM                    TEXT_DESC
C                   PARM                    PUB_AUT
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   EVAL      API_NAME = 'QSZCRTPL'
C                   EXSR      ERRCOD
C                   ENDIF
C*
C* Setup for Product Load for MRI Objects
C* Fill Product Load Information Parameter
C*
C                   EVAL      QSZLT = '*LNG'
C                   EVAL      QSZLID = '2924'
C*
C* Fill Principal Library Information Parameter
C*
C                   EVAL      QSZPEP = 'ABCPGMMRI2'
C*
C* Fill Preoperation Exit Programs Parameter
C*
C                   EVAL      QSZPEP00 = 'ABCPGMMRI1'
C*
C* Fill Additional Library List Parameter
C*       None
C*
C* Fill Folder List Parameter
C*       None
C*
C* Let's create the product load for the ABC Product - MRI Objects
C*
C                   CALL      'QSZCRTPL'
C                   PARM      'ABC0029'     PROD_ID_NM
C                   PARM                    QSZLI
C                   PARM      'ABC2924'     SEC_LANG
C                   PARM                    QSZLI00
C                   PARM                    QSZAL
C                   PARM                    NBR_ADD_LB
C                   PARM                    QSZPE
C                   PARM                    NBR_PE
C                   PARM                    QSZFL
```

```
C                   PARM                    NBR_FLDRS
C                   PARM                    TEXT_DESC
C                   PARM                    PUB_AUT
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   EVAL      API_NAME = 'QSZCRTPL'
C                   EXSR      ERRCOD
C                   ENDIF
C*
C                   ENDSR
C*
C*****************************************************************
C*****************************************************************
C*
C* Subroutine: COBJD
C*
C* Descriptive Name: Change object descriptions for ABC Product.
C*
C* Description: This subroutine will change the object
C*              descriptions for all objects that make up the
C*              ABC Product.  Currently that is 15 objects.  They
C*              are listed at the end of this program.
C*
C*****************************************************************
C*****************************************************************
C*
C     COBJD         BEGSR
C*
C* Need to associate all objects with the ABC Product
C*
C     1             DO        15            I                 3 0
C                   EVAL      OBJ_PTR = %ADDR(OBJ_INFO(I))
C                   EVAL      OBJNAM = OBJ_NAME + 'ABC'
C                   EVAL      LP5 = LP_ID
C                   EVAL      PID13 = PRD_OPT_ID
C                   EVAL      LID12 = PRD_OPT_LD
C                   EVAL      TYPE = OBJ_TYPE
C*
C                   CALL      'QLICOBJD'
C                   PARM                    RTN_LIB          10
C                   PARM                    OBJNAM
C                   PARM                    TYPE             10
C                   PARM                    COBJI
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   EVAL      API_NAME = 'QLICOBJD'
C                   EXSR      ERRCOD
C                   ENDIF
C*
C                   ENDDO
C*
C                   ENDSR
C*
C*****************************************************************
C*****************************************************************
```

```
C*
C* Subroutine: PKGPO
C*
C* Descriptive Name: Package software ABC Product.
C*
C* Description: This subroutine will package the ABC Product.
C*              It makes sure that all objects exist that are
C*              associated with the product.
C*
C*****************************************************************
C*****************************************************************
C*
C     PKGPO         BEGSR
C*
C* Setup for packing the ABC Product.
C* Fill Product Option Information Parameter
C*
C                   EVAL      QSZOPT02 = '0000'
C                   EVAL      QSZPID01 = PROD_ID
C                   EVAL      QSZRL01 = RLS_LVL
C                   EVAL      QSZLID00 = '*ALL'
C                   EVAL      QSZERVED03 = *BLANKS
C*
C* Let's package the ABC Product.
C*
C*
C                   CALL      'QSZPKGPO'
C                   PARM                    QSZPOI
C                   PARM      '*YES'        REPKG             4
C                   PARM      '*NO'         ALWCHG            5
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   EVAL      API_NAME = 'QSZPKGPO'
C                   EXSR      ERRCOD
C                   ENDIF
C*
C                   ENDSR
C*
C*****************************************************************
C*****************************************************************
C*
C* Subroutine: ERROR
C*
C* Descriptive Name: Process API errors.
C*
C* Description: This subroutine will print a line to a spooled
C*              file if any errors are returned in the error code
C*              parameter.
C*
C*****************************************************************
C*****************************************************************
C*
C     ERRCOD        BEGSR
C*
C* Process errors returned from the API.
C*
C* If first error found, then open QPRINT *PRTF
C*
C                   IF        FIRST_ERR = '0'
C                   OPEN      QPRINT
C                   EVAL      FIRST_ERR = '1'
```

```
     C                   ENDIF
     C*
     C* Output the error and the API that received the error
     C*
     C                   EXCEPT    BAD_NEWS
     C*
     C                   ENDSR
     OQPRINT    E            BAD_NEWS       1
     O                                           'Failed in API '
     O                       API_NAME
     O                                           'with error '
     O                       QUSEI
**CTDATA OBJ_INFO
ABCPGMMRM1*PGM      000050010ABCABCV3R1M0
ABCPGMMRM2*PGM      000050010ABCABCV3R1M0
ABCPGMMRI1*PGM      000029240ABCABCV3R1M0
ABCPGMMRI2*PGM      000029240ABCABCV3R1M0
ABCPGM    *PGM      000050010ABCABCV3R1M0
QCLSRC    *FILE     000029240ABCABCV3R1M0
ABCDSPF   *FILE     000029240ABCABCV3R1M0
ABCPF     *FILE     000029240ABCABCV3R1M0
ABCMSG    *MSGF     000029240ABCABCV3R1M0
ABC       *CMD      000029240ABCABCV3R1M0
ABCPNLGRP *PNLGRP   000029240ABCABCV3R1M0
ABC0050   *PRDDFN   000050010ABCABCV3R1M0
ABC0050   *PRDLOD   000050010ABCABCV3R1M0
ABC0029   *PRDLOD   000029240ABCABCV3R1M0
ABC       *LIB      000050010ABCABCV3R1M0
```

Related reference:

"Example in OPM RPG: Packaging a product" on page 240
This OPM RPG program creates objects 12 through 14 and packages your product.

Examples: Retrieving a file description to a user space

These examples show a high-level language program that uses a user space as a receiver variable by retrieving a file description to a user space. Use this approach only when your high-level language supports pointers.

The program accepts the following parameters:

- User space name and library
- File name and library
- Record format

Here is the sequence of steps to retrieve a file description to a user space:

1. The program creates a user space to store the data in, changes the user space to be automatically extendable, and retrieves a pointer to the user space.
2. The program calls the Retrieve File Description API to retrieve the file definition template and uses the user space as the receiver variable.

These examples use an automatically extended user space as the receiver variable on a retrieve API. A user space can return a varying amount of information depending on the file description being retrieved. The user space is automatically extended up to 16MB to accommodate the information being retrieved.

Related reference:

Retrieve Database File Description (QDBRTVFD) API

Example in ILE C: Retrieving a file description to a user space:

This ILE C program creates a user space, changes the user space to be automatically extendable, and retrieves the file description to the user space using pointers.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*****************************************************************/
/* Program Name:          RTVFD                                  */
/*                                                               */
/* Program Language:      ILE C                                  */
/*                                                               */
/* Description:           Retrieve a file definition template to a  */
/*                        user space.                            */
/*                                                               */
/* Header Files Included: <stdlib.h>                             */
/*                        <signal.h>                             */
/*                        <string.h>                             */
/*                        <stdio.h>                              */
/*                        <quscrtus.h>                           */
/*                        <quscusat.h>                           */
/*                        <qusptrus.h>                           */
/*                        <qdbrtvfd.h>                           */
/*                        <qusec.h>                              */
/*                        <qus.h>                                */
/*                        <qliept.h>                             */
/*                                                               */
/* APIs Used:             QUSCRTUS - Create User Space           */
/*                        QUSCUSAT - Change User Space Attributes   */
/*                        QUSPTRUS - Retrieve Pointer to User Space */
/*                        QDBRTVFD - Retrieve File Description      */
/*****************************************************************/
 #include <stdlib.h>
 #include <signal.h>
 #include <string.h>
 #include <stdio.h>
 #include <quscrtus.h>
 #include <quscusat.h>
 #include <qusptrus.h>
 #include <qdbrtvfd.h>
 #include <qusec.h>
 #include <qus.h>
 #include <qliept.h>             /* Note that this must be the last */
                                 /* include specified.              */

 int error_flag = 0;             /* Set by error handler            */
/*****************************************************************/
/* Function:    error_handler                                    */
/* Description: Handle exceptions.                               */
/*****************************************************************/

void error_handler(int errparm)
{
 _INTRPT_Hndlr_Parms_T ExcDta = {0};

 _GetExcData(&ExcDta);
 error_flag = 1;
 signal(SIGALL,error_handler);
}

/*****************************************************************/
/* Start of main procedure                                       */
/*****************************************************************/

 main(int argc, char **argv)
{
  typedef struct attrib_struct {
    int attrib_count;
    Qus_Vlen_Rec_3_t  keyinfo;
    char key_value;
  } attrib_struct;
```

```
Qus_EC_t error_code;              /* Error code parameter        */
attrib_struct attrib_info;        /* Attribute to change         */
char user_space[21];              /* User space and library      */
char descr[50];                   /* Text description            */
char initial_value = 0x00;        /* Initial value for user space*/
char return_lib[10];              /* Return library              */
char ret_file_lib[20];            /* Returned file and library   */
char file_and_lib[21];            /* File and library            */
char record_fmt[11];              /* Record format name          */
char *space_ptr;                  /* Pointer to user space object*/

/*****************************************************************/
/* Start of executable code.                                     */
/*****************************************************************/
if (argc != 4) {
   printf("This program requires 3 parameters:\n");
   printf("   1) User space name and library\n");
   printf("   2) File name and library\n");
   printf("   3) Record format name\n");
   printf("Please retry with those parameters.\n");
   exit(1);
}

memcpy(user_space, *++argv, 20);
memcpy(file_and_lib, *++argv, 20);
memcpy(record_fmt, *++argv, 10);
memset(desc,' ',50);
memcpy(descr,"RTVFD User Space",16);

signal(SIGALL,error_handler);     /* Enable the error handler    */
error_code.Bytes_Provided=0;      /* Have APIs return exceptions */

/*****************************************************************/
/* Create the user space.                                        */
/*****************************************************************/
QUSCRTUS(user_space,              /* User space                  */
         "          ",            /* Extended attribute          */
         1024,                    /* Initial size                */
         &initial_value,          /* Initial value               */
         "*CHANGE   ",            /* Public authority            */
         descr,                   /* Text description            */
         "*YES      ",            /* Replace if it exists        */
         &error_code,             /* Error code                  */
         "*USER     ");           /* Domain = USER               */

if (error_flag) {
   exit(1);
}

/*****************************************************************/
/* Initialize the attributes to change structure.                */
/*****************************************************************/
attrib_info.attrib_count = 1;     /* Number of attributes        */
attrib_info.keyinfo.Key = 3;      /* Key of attribute to change  */
attrib_info.keyinfo.Length_Vlen_Record = 1;
                                  /* Length of data              */
attrib_info.key_value='1';        /* Autoextend space            */

/*****************************************************************/
/* Change the user space to be automatically extendable.         */
/*****************************************************************/
QUSCUSAT(return_lib,              /* Return library              */
         user_space,              /* User space name and library */
         &attrib_info,            /* Attributes to change        */
         &error_code);            /* Error code                  */
```

```
  if (error_flag) {
     exit(1);
  }

  /*****************************************************************/
  /* Retrieve a pointer to the user space object.                  */
  /*****************************************************************/
  QUSPTRUS(user_space,&space_ptr);

  if (error_flag) {
     exit(1);
  }

  /*****************************************************************/
  /* Retrieve the file description information to the user space.  */
  /*****************************************************************/
  QDBRTVFD(space_ptr,               /* Receiver variable           */
           16776704,                /* Return up to 16MB minus 512 */
                                    /* bytes of data               */
           ret_file_lib,            /* Returned file and library   */
           "FILD0100",              /* File definition template    */
           file_and_lib,            /* File and library name       */
           record_fmt,              /* Record format name          */
           "0",                     /* No override processing      */
           "*LCL      ",            /* Local system                */
           "*INT      ",            /* Internal formats  (1)       */
           &error_code);            /* Error code                  */

  if (error_flag) {
     exit(1);
  }

 }
```

The program uses the value *INT (**(1)**). A description and examples of the internal (*INT) and external (*EXT) formats are provided in the Retrieve Database File Description (QDBRTVFD) API.

Related reference:

"Example in ILE COBOL: Retrieving a file description to a user space"
This ILE COBOL program creates a user space, changes the user space to be automatically extendable, and retrieves the file description to the user space using pointers.

"Example in ILE RPG: Retrieving a file description to a user space" on page 273
This ILE RPG program creates a user space, changes the user space to be automatically extendable, and retrieves the file description to the user space using pointers.

Example in ILE COBOL: Retrieving a file description to a user space:

This ILE COBOL program creates a user space, changes the user space to be automatically extendable, and retrieves the file description to the user space using pointers.

The following program also works with OPM COBOL.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
       IDENTIFICATION DIVISION.
      *****************************************************************
      *****************************************************************
      *
      *  Program:      RTVFD
      *
      *  Language:     COBOL
      *
      *  Description:  This program retrieves a file definition
```

```
*                template to a user space.
*
*  APIs Used:    QDBRTVFD - Retrieve File Description
*                QUSCRTUS - Create User Space
*                QUSCUSAT - Change User Space Attributes
*                QUSPTRUS - Retrieve a pointer to a User Space
*
*****************************************************************
*****************************************************************
 PROGRAM-ID. RTVFD.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
* Error Code parameter include.  As this sample program
* uses COPY to include the error code structure, only the first
* 16 bytes of the error code structure are available.  If the
* application program needs to access the variable length
* exception data for the error, the developer should physically
* copy the QSYSINC include and modify the copied include to
* define additional storage for the exception data.
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Misc. elements
*
 01  MISC.
     05  EXIT-POINT-NAME PIC  X(20) VALUE "EXAMPLE_EXIT_POINT".
     05  EXIT-PGM-NBR    PIC S9(09) VALUE -1 BINARY.
     05  EXIT-PARAMETERS PIC  X(10).
     05  FILE-USED       PIC  X(20).
     05  LIBRARY-NAME    PIC  X(10).
     05  SPACE-SIZE      PIC S9(09) BINARY.
     05  SPACE-INIT      PIC  X(01) VALUE "X'00'".
     05  SPACE-POINTER   POINTER.
     05  FORMAT-NAME-1   PIC  X(08).
     05  OVERRIDES       PIC  X(01) VALUE "0".
     05  SYSTEM          PIC  X(10) VALUE "*LCL".
     05  FORMAT-1        PIC  X(10) VALUE "*INT".
     05  EXT-ATTR        PIC  X(10).
     05  SPACE-AUT       PIC  X(10) VALUE "*CHANGE".
     05  SPACE-TEXT      PIC  X(50) VALUE "QDBRTVFD".
     05  SPACE-REPLACE   PIC  X(10) VALUE "*YES".
     05  SPACE-DOMAIN    PIC  X(10) VALUE "*USER".
     05  API-NAME        PIC  X(10).
 01  CHG-US-ATTR.
     05  NBR-OF-ATTR     PIC S9(09) VALUE 1 BINARY.
     05  ATTR-KEY        PIC S9(09) VALUE 3 BINARY.
     05  DATA-SIZE       PIC S9(09) VALUE 1 BINARY.
     05  ATTR-DATA       PIC  X(01) VALUE "1".
*
 LINKAGE SECTION.
 01  SPACE-NAME          PIC  X(20).
 01  FILE-NAME           PIC  X(20).
 01  FORMAT-NAME-PARM    PIC  X(10).
*
* Retrieve File Description API include.
*
 COPY QDBRTVFD OF QSYSINC-QLBLSRC.
*
* Beginning of mainline
*
 PROCEDURE DIVISION USING SPACE-NAME, FILE-NAME,
                          FORMAT-NAME-PARM.
 MAIN-LINE.
*
     PERFORM INITIALIZE-SPACE.
     PERFORM PROCESS-SPACE.
```

```
     PERFORM PROGRAM-DONE.
*
* Start of subroutines
*
****************************************************************
 PROCESS-SPACE.
*
* The template returned from QDBRTVFD is now addressable by way
* of SPACE-POINTER; as an example the program will now display
* the access method for the file:
*
     DISPLAY QDBFPACT OF QDB-QDBFH.
*
****************************************************************
*
 INITIALIZE-SPACE.
*
* One time initialization code for this program
*
* Set Error Code structure to not use exceptions
*
     MOVE 16 TO BYTES-PROVIDED OF QUS-EC.
*
* Create a User Space for QDBRTVFD
*
     MOVE 1024 TO SPACE-SIZE.
     CALL "QUSCRTUS" USING SPACE-NAME, EXT-ATTR, SPACE-SIZE,
                           SPACE-INIT, SPACE-AUT, SPACE-TEXT,
                           SPACE-REPLACE, QUS-EC, SPACE-DOMAIN.
*
* Check for errors on QUSCRTUS
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                        MOVE "QUSCRTUS" TO API-NAME,
                        PERFORM API-ERROR-FOUND.
*
* Change the User Space so that it is extendable
*
     CALL "QUSCUSAT" USING LIBRARY-NAME, SPACE-NAME,
                           CHG-US-ATTR, QUS-EC.
*
* Check for errors on QUSCUSAT
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                        MOVE "QUSCUSAT" TO API-NAME,
                        PERFORM API-ERROR-FOUND.
*
* Get a resolved pointer to the User Space
*
     CALL "QUSPTRUS" USING SPACE-NAME, SPACE-POINTER, QUS-EC.
*
* Check for errors on QUSPTRUS
*
     IF BYTES-AVAILABLE OF QUS-EC > 0
                        MOVE "QUSPTRAT" TO API-NAME,
                        PERFORM API-ERROR-FOUND.
*
* If no errors, then call QDBRTVFD passing the address of the
* User Space as the receiver variable.  To accomplish this,
* assign the address of QDB-QDBFH to SPACE-POINTER and then
* pass QDB-QDBFH.
*
     SET ADDRESS OF QDB-QDBFH TO SPACE-POINTER.
*
     MOVE 16776704 TO SPACE-SIZE.
     MOVE "FILD0100" TO FORMAT-NAME-1.
*
```

```
     CALL "QDBRTVFD" USING QDB-QDBFH, SPACE-SIZE, FILE-USED,
                           FORMAT-NAME-1, FILE-NAME,
                           FORMAT-NAME-PARM, OVERRIDES,
                           SYSTEM OF MISC, FORMAT-1, QUS-EC.
 *
 * Check for errors on QDBRTVFD
 *
     IF BYTES-AVAILABLE OF QUS-EC > 0
                        MOVE "QDBRTVFD" TO API-NAME,
                        PERFORM API-ERROR-FOUND.
 ****************************************************************
  API-ERROR-FOUND.
 *
 * Log any error encountered, and exit the program
 *
     DISPLAY API-NAME.
     DISPLAY EXCEPTION-ID OF QUS-EC.
     PERFORM PROGRAM-DONE.
 ****************************************************************
  PROGRAM-DONE.
 *
 * Exit the program
 *
     STOP RUN.
```

Related reference:

"Example in ILE C: Retrieving a file description to a user space" on page 267
This ILE C program creates a user space, changes the user space to be automatically extendable, and retrieves the file description to the user space using pointers.

Example in ILE RPG: Retrieving a file description to a user space:

This ILE RPG program creates a user space, changes the user space to be automatically extendable, and retrieves the file description to the user space using pointers.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
D*************************************************************
D*************************************************************
D*
D*  Program:      RTVFD
D*
D*  Language:     ILE RPG
D*
D*  Description:  This program retrieves a file definition
D*                template to a user space.
D*
D*  APIs Used:    QDBRTVFD - Retrieve File Description
D*                QUSCRTUS - Create User Space
D*                QUSCUSAT - Change User Space Attributes
D*                QUSPTRUS - Retrieve a pointer to a User Space
D*
D*************************************************************
D*************************************************************
D*
D* Error Code parameter include
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D* Not shown due to its size, this program also includes QDBRTVFD
D* and defines all of the data structures in QDBRTVFD as being
D* BASED(SPCPTR).  For illustrative purposes, this sample shows
D* only the first significant data structure.
D*
D*************************************************************
```

```
D*
D*File Definition Template (FDT) Header
D*
D*****************************************************************
D*This section is always located at the beginning of the
D*returned data.
D*****************************************************************
DQDBQ25           DS                  BASED(SPCPTR)
D*                                     Header information - The
D*                                     FDT starts here
D QDBFYRET                1      4B 0
D*                                     Bytes returned - The length
D*                                     of the data returned
D QDBFYAVL                5      8B 0
D*                                     Bytes available - The number
D*                                     of bytes provided for the
D*                                     file definition template
D*                                     data
D*QDBFHFLG                            2
D  QDBBITS27              9     10
D*  QDBRSV100             2           BITS
D*  QDBFHFPL00            1           BIT
D*  QDBRSV200             1           BIT
D*  QDBFHFSU00            1           BIT
D*  QDBRSV300             1           BIT
D*  QDBFHFKY00            1           BIT
D*  QDBRSV400             1           BIT
D*  QDBFHFLC00            1           BIT
D*  QDBFKFSO00            1           BIT
D*  QDBRSV500             1           BIT
D*  QDBFHSHR00            1           BIT
D*  QDBRSV600             2           BITS
D*  QDBFIGCD00            1           BIT
D*  QDBFIGCL00            1           BIT
D*                                     Attribute Bytes
D QDBRSV7                11     14
D*                                     Reserved.
D QDBLBNUM               15     16B 0
D*                                     Number Of Data Members
D*                                     1 = Externally described
D*                                         physical file, or program
D*                                         described physical file
D*                                         that is NOT linked to a
D*                                         Data Dictionary.
D*                                     1-32 = Number of Data
D*                                            Dictionary record
D*                                            formats for a program
D*                                            described physical
D*                                            file that is linked to
D*                                            a Data Dictionary.
D*                                     1-32 = Number of based-on
D*                                            physical files for
D*                                            a logical file.
D*QDBFKDAT                           14
D  QDBFKNUM00            17     18B 0
D  QDBFKMXL00            19     20B 0
D* QDBFKFLG00                         1
D   QDBBITS28            21     21
D*  QDBRSV802             1           BIT
D*  QDBFKFCS02            1           BIT
D*  QDBRSV902             4           BITS
D*  QDBFKFRC02            1           BIT
D*  QDBFKFLT02            1           BIT
D  QDBFKFDM00            22     22
D  QDBRSV1000            23     30
D*                                     Keyed Sequence Access Path
D QDBFHAUT               31     40
```

```
D*                                Public Authority (AUT)
D*                                '*CHANGE   ' = Public change
D*                                               authority.
D*                                '*ALL      ' = Public all
D*                                               authority.
D*                                '*USE      ' = Public use
D*                                               authority.
D*                                '*EXCLUDE  ' = Public exclude
D*                                               authority.
D*                                'authorization-list-name'
D*                                             = Name of the
D*                                               authorization
D*                                               list whose
D*                                               authority is
D*                                               used for the
D*                                               file.
D*                                This is the original public
D*                                authority that the file was
D*                                created with, NOT the current
D*                                public authority for the file.
D QDBFHUPL              41     41
D*                                Preferred Storage Unit (UNIT)
D*                                X'00' = The storage space for
D*                                        the file and its
D*                                        members can be
D*                                        allocated on any
D*                                        available auxiliary
D*                                        storage unit (*ANY).
D*                                X'01'-X'FF' = The unit
D*                                              identifier (a
D*                                              number from 1
D*                                              to 255 assigned
D*                                              when the disk
D*                                              device is
D*                                              configured) of
D*                                              a specific
D*                                              auxiliary
D*                                              storage unit on
D*                                              the system.
D QDBFHMXM              42     43B 0
D*                                Maximum Members (MAXMBRS)
D*                                0 = No maximum is specified
D*                                    for the number of members,
D*                                    the system maximum of
D*                                    32,767 members is used
D*                                    (*NOMAX).
D*                                1-32,767 = The value for the
D*                                           maximum number of
D*                                           members that the
D*                                           file can have
D*                                           (maximum-members).
D QDBFWTFI              44     45B 0
D*                                Maximum File Wait Time
D*                                (WAITFILE)
D*                                -1 = The default wait time
D*                                     specified in the class
D*                                     description is used as
D*                                     the wait time for the
D*                                     file (*CLS).
D*                                0 = A program does NOT wait
D*                                    for the file, an
D*                                    immediate allocation of
D*                                    the file is required
D*                                    (*IMMED).
D*                                1-32,767 = The number of
D*                                           seconds that a
D*                                           program waits for
```

```
D*                                        the file (number-
D*                                        of-seconds).
D QDBFHFRT               46     47B 0
D*                                 Records To Force A Write
D*                                 (FRCRATIO)
D*                                 0 = There is NO force write
D*                                     ratio, the system
D*                                     determines when the
D*                                     records are written to
D*                                     auxiliary storage (*NONE).
D*                                 1-32,767 = The number of
D*                                            inserted, updated,
D*                                            or deleted records
D*                                            that are processed
D*                                            before they are
D*                                            explicitly forced
D*                                            to auxiliary
D*                                            storage (number-
D*                                            of-records-before-
D*                                            force).
D QDBHMNUM               48     49B 0
D*                                 Number Of Members
D*                                 0-32,767 = The current number
D*                                            of members for the
D*                                            file.
D QDBRSV11               50     58
D*                                 Reserved.
D QDBFBRWT               59     60B 0
D*                                 Maximum Record Wait Time
D*                                 (WAITRCD)
D*                                 -2 = The wait time is the
D*                                      maximum allowed by the
D*                                      system, 32,767 seconds
D*                                      (*NOMAX).
D*                                 -1 = A program does NOT wait
D*                                      for the record, an
D*                                      immediate allocation of
D*                                      the record is required
D*                                      (*IMMED).
D*                                 1-32,767 = The number of
D*                                            seconds that a
D*                                            program waits for
D*                                            the record
D*                                            (number-of-
D*                                            seconds).
D*QDBQAAF00                         1
D  QDBBITS29             61     61
D*  QDBRSV1200            7           BITS
D*  QDBFPGMD00            1           BIT
D*                                 Additional Attribute Flags
D QDBMTNUM               62     63B 0
D*                                 Total Number Of Record
D*                                 Formats
D*                                 1-32 = Number of record
D*                                        formats for the file.
D*QDBFHFL2                          2
D  QDBBITS30             64     65
D*  QDBFJNAP00            1           BIT
D*  QDBRSV1300            1           BIT
D*  QDBFRDCP00            1           BIT
D*  QDBFWTCP00            1           BIT
D*  QDBFUPCP00            1           BIT
D*  QDBFDLCP00            1           BIT
D*  QDBRSV1400            9           BITS
D*  QDBFKFND00            1           BIT
D*                                 Additional Attribute Flags
D QDBFVRM                66     67B 0
```

```
D*                                First Supported
D*                                Version Release Modification
D*                                Level
D*                                X'0000' = Pre-Version 2
D*                                          Release 1
D*                                          Modification 0 file.
D*                                X'1500' = Version 2 Release 1
D*                                          Modification 0,
D*                                          V2R1M0, file.
D*                                X'1501' = Version 2 Release 1
D*                                          Modification 1,
D*                                          V2R1M1, file.
D*                                X'1600' = Version 2 Release 2
D*                                          Modification 0,
D*                                          V2R2M0, file.
D*                                New Database support is used
D*                                in the file which will
D*                                prevent it from being saved
D*                                and restored to a prior
D*                                Version Release and
D*                                Modification level.
D*QDBQAAF2                        1
D  QDBBITS31             68     68
D*  QDBFHMCS00            1           BIT
D*  QDBRSV1500            1           BIT
D*  QDBFKNLL00            1           BIT
D*  QDBFNFLD00            1           BIT
D*  QDBFVFLD00            1           BIT
D*  QDBFTFLD00            1           BIT
D*  QDBFGRPH00            1           BIT
D*  QDBRSV1600            1           BIT
D*                                Additional Attribute Flags
D QDBRSV17               69     69
D*                                Reserved.
D QDBFHCRT               70     82
D*                                File Level Identifier
D*                                The date of the file in
D*                                internal standard format
D*                                (ISF), CYYMMDDHHMMSS.
D*QDBFHTX                        52
D  QDBRSV1800            83     84
D  QDBFHTXT00            85    134
D*                                File Text Description
D QDBRSV19               135   147
D*                                Reserved
D*QDBFSRC                        30
D  QDBFSRCF00            148   157
D  QDBFSRCM00            158   167
D  QDBFSRCL00            168   177
D*                                Source File Fields
D QDBFKRCV               178   178
D*                                Access Path Recovery
D*                                (RECOVER)
D*                                'A' = The file has its access
D*                                      path built after the
D*                                      IPL has been completed
D*                                      (*AFTIPL).
D*                                'N' = The access path of the
D*                                      file is NOT built
D*                                      during or after an IPL
D*                                      (*NO).  The file's
D*                                      access path is built
D*                                      when the file is next
D*                                      opened.
D*                                'S' = The file has its access
D*                                      path built during the
D*                                      IPL (*IPL).
```

```
D QDBRSV20              179    201
D*                             Reserved.
D QDBFTCID              202    203B 0
D*                             Coded Character Set
D*                             Identifier, CCSID, For
D*                             Text Description (TEXT)
D*                             0 = There is NO text
D*                                 description for the file.
D*                             1-65,535 = The CCSID for the
D*                                        file's text
D*                                        description.
D QDBFASP               204    205
D*                             Auxiliary Storage Pool (ASP)
D*                             X'0000' = The file is
D*                                       located on the
D*                                       system auxiliary
D*                                       storage pool.
D*                             X'0002'-X'0010' = The user
D*                                       auxiliary storage
D*                                       pool the file is
D*                                       located on
D*                                       (asp-identifier).
D QDBRSV21              206    206
D*                             Reserved.
D QDBXFNUM              207    208B 0
D*                             Maximum Number Of Fields
D*                             1-8000 = The number of fields
D*                                      in the file's record
D*                                      format that contains
D*                                      the largest number
D*                                      of fields.
D QDBRSV22              209    284
D*                             Reserved.
D QDBFODIC              285    288B 0
D*                             Offset from the start of the
D*                             FDT header, Qdbfh, to the
D*                             IDDU/SQL Data Dictionary
D*                             Area, Qdbfdic.
D QDBRSV23              289    302
D*                             Reserved.
D QDBFFIGL              303    304B 0
D*                             File Generic Key Length
D*                             0-2000 = The length of the
D*                                      key before the first
D*                                      *NONE key field for
D*                                      the file.
D*                             If this file has an arrival
D*                             sequence access path, this
D*                             field is NOT applicable.
D QDBFMXRL              305    306B 0
D*                             Maximum Record Length
D*                             1-32766 = The length of the
D*                                       record in the
D*                                       file's record
D*                                       format that
D*                                       contains the
D*                                       largest number of
D*                                       bytes.
D QDBRSV24              307    314
D*                             Reserved.
D QDBFGKCT              315    316B 0
D*                             File Generic Key Field Count
D*                             0-120 = The count of the
D*                                     number of key fields
D*                                     before the first
D*                                     *NONE key field for
D*                                     the file.
```

```
D*                                 If this file has an arrival
D*                                 sequence access path, this
D*                                 field is NOT applicable.
D QDBFOS                  317    320B 0
D*                                 Offset from the start of the
D*                                 FDT header, Qdbfh, to the
D*                                 File Scope Array, Qdbfb.
D QDBRSV25                321    328
D*                                 Reserved.
D QDBFOCS                 329    332B 0
D*                                 Offset from the start of the
D*                                 FDT header, Qdbfh, to the
D*                                 Alternative Collating
D*                                 Sequence Table section,
D*                                 Qdbfacs.
D QDBRSV26                333    336
D*                                 Reserved.
D QDBFPACT                337    338
D*                                 Access Path Type
D*                                 'AR' = Arrival sequence
D*                                        access path.
D*                                 'KC' = Keyed sequence access
D*                                        path with duplicate
D*                                        keys allowed.
D*                                        Duplicate keys are
D*                                        accessed in first-
D*                                        changed-first-out
D*                                        (FCFO) order.
D*                                 'KF' = Keyed sequence access
D*                                        path with duplicate
D*                                        keys allowed.
D*                                        Duplicate keys are
D*                                        accessed in first-
D*                                        in-first-out
D*                                        (FIFO) order.
D*                                 'KL' = Keyed sequence access
D*                                        path with duplicate
D*                                        keys allowed.
D*                                        Duplicate keys are
D*                                        accessed in last-
D*                                        in-first-out
D*                                        (LIFO) order.
D*                                 'KN' = Keyed sequence access
D*                                        path with duplicate
D*                                        keys allowed.
D*                                        No order is guaranteed
D*                                        when accessing
D*                                        duplicate keys.
D*                                        Duplicate keys are
D*                                        accessed in one of the
D*                                        following methods:
D*                                        (FCFO) (FIFO) (LIFO).
D*                                 'KU' = Keyed sequence access
D*                                        path with NO duplicate
D*                                        keys allowed (UNIQUE).
D QDBFHRLS                339    344
D*                                 File Version Release
D*                                 Modification Level
D*                                 'VxRyMz' = Where x is the
D*                                            Version, y is the
D*                                            Release, and z is
D*                                            the Modification
D*                                            level
D*                                            example V2R1M1
D*                                            Version 2 Release
D*                                            1 Modification 1
D QDBRSV27                345    364
```

```
D*                                  Reserved.
D QDBPFOF               365    368B 0
D*                                  Offset from the start of the
D*                                  FDT header, Qdbfh, to the
D*                                  Physical File Specific
D*                                  Attributes section, Qdbfphys.
D QDBLFOF               369    372B 0
D*                                  Offset from the start of the
D*                                  FDT header, Qdbfh, to the
D*                                  Logical File Specific
D*                                  Attributes section, Qdbflogl.
D*QDBFSSFP00                       6
D* QDBFNLSB01                       1
D   QDBBITS58           373    373
D*  QDBFSSCS02             3           BITS
D*  QDBR10302              5           BITS
D  QDBFLANG01           374    376
D  QDBFCNTY01           377    378
D*                                  Sort Sequence Table
D QDBFJORN              379    382B 0
D*                                  Offset from the start of the
D*                                  FDT header, Qdbfh, to the
D*                                  Journal Section, Qdbfjoal.
D QDBRSV28              383    400
D*                                  Reserved.
D*****************************************************************
D*
D*The FDT header ends here.
D*
D*****************************************************************
D*
D* Misc. elements
D*
DSPC_NAME         S             20
DFILE_NAME        S             20
DFMT_NAME         S             10
DFILE_USED        S             20
DLIB_NAME         S             10
DSPC_SIZE         S              9B 0
DSPC_INIT         S              1    INZ(X'00')
DSPCPTR           S               *
DFORMAT           S              8
DOVERRIDES        S              1    INZ('0')
DSYSTEM           S             10    INZ('*LCL')
DFORMAT_1         S             10    INZ('*INT')
DCHG_ATTR         DS
D NBR_ATTR                       9B 0 INZ(1)
D ATTR_KEY                       9B 0 INZ(3)
D DATA_SIZE                      9B 0 INZ(1)
D ATTR_DATA                      1    INZ('1')
C*
C* Start of mainline
C*
C     *ENTRY        PLIST
C                   PARM                    SPC_NAME
C                   PARM                    FILE_NAME
C                   PARM                    FMT_NAME
C*
C                   EXSR      INIT
C                   EXSR      PROCES
C                   EXSR      DONE
C*
C* Start of subroutines
C*
C*****************************************************************
C     PROCES        BEGSR
C*
```

```
C* The template returned from QDBRTVFD is now addressable by way
C* of SPCPTR; as an example the program will now display the
C* access method for the file:
C*
C                   DSPLY                   QDBFPACT
C                   ENDSR
C*
C*****************************************************************
C     INIT          BEGSR
C*
C* One time initialization code for this program
C*
C* Set Error Code structure to not use exceptions
C*
C                   Z-ADD     16            QUSBPRV
C*
C* Create a User Space for QDBRTVFD
C*
C                   CALL      'QUSCRTUS'
C                   PARM                    SPC_NAME
C                   PARM      *BLANKS       EXT_ATTR         10
C                   PARM      1024          SPC_SIZE
C                   PARM                    SPC_INIT
C                   PARM      '*CHANGE'     SPC_AUT          10
C                   PARM      'QDBRTVFD'    SPC_TEXT         50
C                   PARM      '*YES'        SPC_REPLAC       10
C                   PARM                    QUSEC
C                   PARM      '*USER'       SPC_DOMAIN       10
C*
C* Check for errors on QUSCRTUS
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QUSCRTUS'    APINAM           10
C                   EXSR      APIERR
C                   END
C*
C* Change the User Space so that it is extendable
C*
C                   CALL      'QUSCUSAT'
C                   PARM                    LIB_NAME
C                   PARM                    SPC_NAME
C                   PARM                    CHG_ATTR
C                   PARM                    QUSEC
C*
C* Check for errors on QUSCUSAT
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QUSCUSAT'    APINAM           10
C                   EXSR      APIERR
C                   END
C*
C* Get a resolved pointer to the User Space
C*
C                   CALL      'QUSPTRUS'
C                   PARM                    SPC_NAME
C                   PARM                    SPCPTR
C                   PARM                    QUSEC
C*
C* Check for errors on QUSPTRUS
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QUSPTRUS'    APINAM           10
C                   EXSR      APIERR
C                   END
C*
C* If no errors, then call QDBRTVFD passing the address of the
C* User Space as the receiver variable.  As Data Structure
```

```
C* QDBQ25 is defined as BASED(SPCPTR) and SPCPTR is set to the
C* first byte of the User Space, simply passing QDBQ25 will cause
C* QDBRTVFD to use the User Space.
C*
C                   CALL      'QDBRTVFD'
C                   PARM                    QDBQ25
C                   PARM      16776704      SPC_SIZE
C                   PARM                    FILE_USED
C                   PARM      'FILD0100'    FORMAT
C                   PARM                    FILE_NAME
C                   PARM                    FMT_NAME
C                   PARM                    OVERRIDES
C                   PARM                    SYSTEM
C                   PARM                    FORMAT_1
C                   PARM                    QUSEC
C*
C* Check for errors on QDBRTVFD
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QDBRTVFD'    APINAM          10
C                   EXSR      APIERR
C                   END
C                   ENDSR
C*****************************************************************
C     APIERR        BEGSR
C*
C* Log any error encountered, and exit the program
C*
C     APINAM        DSPLY
C     QUSEI         DSPLY
C                   EXSR      DONE
C                   ENDSR
C*****************************************************************
C     DONE          BEGSR
C*
C* Exit the program
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C                   ENDSR
```

Related reference:

"Example in ILE C: Retrieving a file description to a user space" on page 267
This ILE C program creates a user space, changes the user space to be automatically extendable, and retrieves the file description to the user space using pointers.

Examples: Using data queues or user queues

Both data queues and user queues provide a means for one or more processes to communicate asynchronously. Both queues can be processed by first-in first-out (FIFO), last-in first-out (LIFO), or by key.

Related concepts:

"APIs overview" on page 1
This API information describes most of the IBM i APIs and some APIs for related licensed programs that run on the i operating system.

"Domains" on page 114
A *domain* is a characteristic of an object that controls how programs can access the object. All objects are assigned a domain attribute when they are created.

Considerations for using data queues and user queues:

To decide whether to use data queues or user queues, you need to consider your programming experience, the performance of each queue type, and the operations on queue entries.

First, your programming experience is an important consideration in selecting a queue type. If you are familiar with C or MI programming, you might want to select the user queue. User queues can be accessed only through MI, and MI can be used only by ILE RPG, ILE COBOL, C, and MI programs.

Next, performance plays an important part in determining what type of queue to use. As stated in System APIs or CL commands--when to use each, APIs generally give better performance than CL commands. Also, MI instructions perform better than an external call to an API because APIs have overhead associated with them. User queues use MI instructions to manipulate entries; data queues use APIs. Therefore, the user queue has better performance than the data queue.

Last, you need to consider how the queue entries are manipulated. For example, you need a way to perform enqueue and dequeue operations on entries from a queue. As stated earlier, user queues use MI instructions to manipulate entries. Specifically, you use the ENQ MI instruction to enqueue a message, and the DEQ MI instruction to dequeue a message. If you are running at security level 40 or greater, you must ensure that the user queue is created in the user domain in order to directly manipulate a user queue using MI instructions. Because data queue entries are manipulated by APIs, the security level of the machine does not limit the use of the API.

You cannot create a user queue object in a library that does not permit user-domain objects, which is determined by the QALWUSRDMN system value. Data queues are always created in the system domain, so there is no problem with the data queue being created into a specific library.

The following summary can help you select the type of queue that is right for your program:

- Use user queues when:
  - You have a programming background in MI.
  - You need the additional performance of an API for creating and deleting and MI instructions for manipulating entries.
  - You do not need to create a user-domain queue into a library where the QALWUSRDMN system value does not permit user-domain user objects when at security level 40 or 50.
- Use data queues when:
  - You have a programming background in or prefer to program in a high-level language such as COBOL, C, or RPG.
  - You do not need the additional performance of MI instructions for directly manipulating entries.
  - You need to create queues into a library that is not listed in the QALWUSRDMN system value.

Related concepts:

"Domains" on page 114
A *domain* is a characteristic of an object that controls how programs can access the object. All objects are assigned a domain attribute when they are created.

"APIs overview" on page 1
This API information describes most of the IBM i APIs and some APIs for related licensed programs that run on the i operating system.

Example in ILE C: Using data queues:

This ILE C program shows how to use APIs to create and manipulate a data queue.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*                                                                   */
/*Program Name: DQUEUEX                                              */
/*                                                                   */
/*Program Language:  ILE C                                           */
/*                                                                   */
```

```
/*Description: This program illustrates how to use APIs to create    */
/*             and manipulate a data queue.                           */
/*                                                                    */
/*                                                                    */
/*Header Files Included: <stdio.h>                                    */
/*                       <string.h>                                   */
/*                       <stdlib.h>                                   */
/*                       <decimal.h>                                  */
/*                       <qrcvdtaq.h>                                 */
/*                       <qsnddtaq.h>                                 */
/*                                                                    */
/*APIs Used:     QSNDDTAQ - Send data queue                          */
/*               QRCVDTAQ - Receive data queue                       */
/*                                                                    */
/**********************************************************************/
/**********************************************************************/
/**********************************************************************/
/*                            Includes                                */
/**********************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <decimal.h>
#include <qsnddtaq.h>                   /* from QSYSINC/h           */
#include <qrcvdtaq.h>                   /* from QSYSINC/h           */

/**********************************************************************/
/*                                                                    */
/*                               Main                                 */
/*                                                                    */
/**********************************************************************/

void main()
{
  decimal(5,0)  DataLength = 10.0d,
                WaitTime = 0.0d;
  char QueueData[10];

  /********************************************************************/
  /* Create library QUEUELIB.                                         */
  /********************************************************************/

  system("CRTLIB LIB(QUEUELIB)");

  /********************************************************************/
  /* Create a data queue called EXAMPLEQ in library QUEUELIB.  The    */
  /* queue will have a maximum entry length set at 10, and will be    */
  /* FIFO (first-in first-out).                                       */
  /********************************************************************/

  system("CRTDTAQ DTAQ(QUEUELIB/EXAMPLEQ) MAXLEN(10)");

  /********************************************************************/
  /* Send information to the data queue.                              */
  /********************************************************************/

  QSNDDTAQ("EXAMPLEQ  ",                /* Data queue name          */
           "QUEUELIB  ",                /* Queue library name       */
           DataLength,                  /* Length of queue entry    */
           "EXAMPLE   ");               /* Data sent to queue       */

  /********************************************************************/
  /* Receive information from the data queue.                         */
  /********************************************************************/

  QRCVDTAQ("EXAMPLEQ  ",                /* Data queue name          */
```

```
          "QUEUELIB  ",                /* Queue library name       */
          &DataLength,                 /* Length of queue entry    */
          &QueueData,                  /* Data received from queue */
          WaitTime);                   /* Wait time                */

  printf("Queue entry information: %.10s\n", QueueData);

  /*****************************************************************/
  /* Delete the data queue.                                        */
  /*****************************************************************/

  system("DLTDTAQ DTAQ(QUEUELIB/EXAMPLEQ)");

  /*****************************************************************/
  /* Delete the library.                                           */
  /*****************************************************************/

  system("DLTLIB LIB(QUEUELIB)");

}
```

Related reference:

"Example in ILE COBOL: Using data queues"
This ILE COBOL program shows how to use APIs to create and manipulate a data queue.

"Example in OPM RPG: Using data queues" on page 288
This OPM RPG program shows how to use APIs to create and manipulate a data queue.

"Example in ILE RPG: Using data queues" on page 291
This ILE RPG program shows how to use APIs to create and manipulate a data queue.

Example in ILE COBOL: Using data queues:

This ILE COBOL program shows how to use APIs to create and manipulate a data queue.

The following program also works for OPM COBOL.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
       IDENTIFICATION DIVISION.
      ****************************************************************
      ****************************************************************
      *
      *  Program Name: DQUEUEX
      *
      *  Programming Language: COBOL
      *
      *  Description:  This program illustrates how to use APIs to
      *                create and manipulate a *DTAQ.
      *
      *  Header Files Included: QUSEC   - Error Code Parameter
      *                         QCAPCMD - Process Command API
      *
      ****************************************************************
      *
      ****************************************************************
       PROGRAM-ID. DQUEUEX.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT LISTING ASSIGN TO PRINTER-QPRINT
                          ORGANIZATION IS SEQUENTIAL.
       DATA DIVISION.
       FILE SECTION.
       FD  LISTING RECORD CONTAINS 132 CHARACTERS
                   LABEL RECORDS ARE STANDARD
```

```
            DATA RECORD IS LIST-LINE.
 01  LIST-LINE        PIC X(132).
 WORKING-STORAGE SECTION.
*
* Error Code parameter include
*
 COPY QUSEC OF QSYSINC-QLBLSRC.
*
* Process Command API Include
*
 COPY QCAPCMD OF QSYSINC-QLBLSRC.
*
* Command strings
*
 01  CRTLIB PIC X(50) VALUE "CRTLIB QUEUELIB".
 01  DLTLIB PIC X(50) VALUE "DLTLIB QUEUELIB".
 01  CRTDQ  PIC X(50)
               VALUE "CRTDTAQ QUEUELIB/EXAMPLEQ MAXLEN(10)".
 01  DLTDQ  PIC X(50) VALUE "DLTDTAQ QUEUELIB/EXAMPLEQ".
*
* Error message text
*
 01  BAD-NEWS.
     05  TEXT1        PIC X(14) VALUE "Failed in API ".
     05  API-NAME     PIC X(10) VALUE "QCAPCMD".
     05  TEXT2        PIC X(11) VALUE "with error ".
     05  EXCEPTION-ID PIC X(07).
*
* Miscellaneous elements
*
 01  COMMAND-LENGTH  PIC S9(09) VALUE 50 BINARY.
 01  RECEIVER        PIC  X(01).
 01  RECEIVER-LENGTH PIC S9(09) VALUE  0 BINARY.
 01  OPTIONS-SIZE    PIC S9(09) VALUE 20 BINARY.
 01  FORMAT-NAME     PIC  X(08) VALUE "CPOP0100".
 01  FIRST-ERROR     PIC  X(01) VALUE "0".
 01  NAME-OF-QUEUE   PIC  X(10) VALUE "EXAMPLEQ".
 01  NAME-OF-LIBRARY PIC  X(10) VALUE "QUEUELIB".
 01  SIZE-OF-MSG     PIC S9(05) VALUE 10 PACKED-DECIMAL.
 01  WAIT-TIME       PIC S9(05) VALUE 0 PACKED-DECIMAL.
 01  MSG             PIC  X(10) VALUE "EXAMPLE".
 01  MSG-BACK        PIC  X(10).
*
* Beginning of mainline
*
 PROCEDURE DIVISION.
 MAIN-LINE.
*
* Initialize the error code parameter.  To signal exceptions to
* this program by the API, you need to set the bytes provided
* field of the error code to zero.  Because this program has
* exceptions sent back through the error code parameter, it sets
* the bytes provided field to the number of bytes it gives the
* API for the parameter.
*
     MOVE 16 TO BYTES-PROVIDED.
*
* Initialize QCAPCMD options control block for CL processing
*
     MOVE 0 TO COMMAND-PROCESS-TYPE.
     MOVE "0" TO DBCS-DATA-HANDLING.
     MOVE "0" TO PROMPTER-ACTION.
     MOVE "0" TO COMMAND-STRING-SYNTAX.
     MOVE SPACES TO MESSAGE-KEY.
     MOVE LOW-VALUES TO RESERVED OF QCA-PCMD-CPOP0100.
*
* Create library QUEUELIB
```

```
*
     CALL QCAPCMD USING CRTLIB, COMMAND-LENGTH, QCA-PCMD-CPOP0100,
                        OPTIONS-SIZE, FORMAT-NAME, RECEIVER,
                        RECEIVER-LENGTH, RECEIVER-LENGTH, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
*
     IF BYTES-AVAILABLE > 0 PERFORM ERROR-FOUND.
*
* Create a data queue called EXAMPLEQ in library QUEUELIB.  The
* queue will have a maximum entry length set at 10, and will be
* FIFO (first-in first-out).
*
     CALL QCAPCMD USING CRTDQ,  COMMAND-LENGTH, QCA-PCMD-CPOP0100,
                        OPTIONS-SIZE, FORMAT-NAME, RECEIVER,
                        RECEIVER-LENGTH, RECEIVER-LENGTH, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
*
     IF BYTES-AVAILABLE > 0 PERFORM ERROR-FOUND.
*
* Send information to the data queue.
*
     CALL "QSNDDTAQ" USING NAME-OF-QUEUE, NAME-OF-LIBRARY,
                           SIZE-OF-MSG, MSG.
*
* Retrieve information from the data queue.
*
     CALL "QRCVDTAQ" USING NAME-OF-QUEUE, NAME-OF-LIBRARY,
                           SIZE-OF-MSG, MSG-BACK, WAIT-TIME.
*
* Display the returned message
*
     DISPLAY MSG-BACK.
*
* Delete the data queue
*
     CALL QCAPCMD USING DLTDQ,  COMMAND-LENGTH, QCA-PCMD-CPOP0100,
                        OPTIONS-SIZE, FORMAT-NAME, RECEIVER,
                        RECEIVER-LENGTH, RECEIVER-LENGTH, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
*
*
     IF BYTES-AVAILABLE > 0 PERFORM ERROR-FOUND.
*
* Delete the library
*
     CALL QCAPCMD USING DLTLIB, COMMAND-LENGTH, QCA-PCMD-CPOP0100,
                        OPTIONS-SIZE, FORMAT-NAME, RECEIVER,
                        RECEIVER-LENGTH, RECEIVER-LENGTH, QUS-EC.
*
* If an exception occurs, the API returns the exception in the
* error code parameter.  The bytes available field is set to
* zero if no exception occurs and greater than zero if an
* exception does occur.
```

```
*
*
     IF BYTES-AVAILABLE > 0 PERFORM ERROR-FOUND.
*
     STOP RUN.
*
* End of MAINLINE
*
****************************************************************
*
 ERROR-FOUND.
*
* Process errors returned from the API.
*
* If first error found, then open QPRINT *PRTF
*
     IF FIRST-ERROR = "0" OPEN OUTPUT LISTING,
                          MOVE "1" TO FIRST-ERROR.
*
* Print the error and the API that received the error
*
     MOVE EXCEPTION-ID OF QUS-EC TO EXCEPTION-ID OF BAD-NEWS.
     WRITE LIST-LINE FROM BAD-NEWS.
```

Related reference:

"Example in ILE C: Using data queues" on page 283
This ILE C program shows how to use APIs to create and manipulate a data queue.

Example in OPM RPG: Using data queues:

This OPM RPG program shows how to use APIs to create and manipulate a data queue.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F****************************************************************
F****************************************************************
F*
F*  Program Name: DQUEUEX
F*
F*  Programming Language: OPM RPG
F*
F*  Description:  This program illustrates how to use APIs to
F*                create and manipulate a *DTAQ.
F*
F*  Header Files Included: QUSEC   - Error Code Parameter
F*                         QCAPCMD - Process Command API
F*
F****************************************************************
F*
FQPRINT  O   F     132            PRINTER                     UC
F****************************************************************
I*
I* Error Code parameter include
I*
I/COPY QSYSINC/QRPGSRC,QUSEC
I*
I* Process Command API Include
I*
I/COPY QSYSINC/QRPGSRC,QCAPCMD
I*
I* Command strings
I*
I            DS
I I            'CRTLIB LIB(QUEUELIB)'    1  20 CRTLIB
I I            'DLTLIB LIB(QUEUELIB)'   21  40 DLTLIB
```

```
I I            'CRTDTAQ DTAQ(QUEUELI-   41  82 CRTDQ
I              'B/EXAMPLEQ) MAXLEN(1-
I              '0)'
I I            'DLTDTAQ DTAQ(QUEUELI-   83 113 DLTDQ
I              'B/EXAMPLEQ)'
I*
I* Miscellaneous data structure
I*
I            DS
I                                        1 100 CMDSTR
I                                     B 101 1040LENSTR
I I            20                     B 105 1080SIZE
I I            0                      B 10901120RCVSIZ
I I            '0'                      113 113 FSTERR
I                                       114 123 APINAM
C*
C* Beginning of mainline
C*
C* Initialize the error code parameter.  To signal exceptions to
C* this program by the API, you need to set the bytes provided
C* field of the error code to zero.  Because this program has
C* exceptions sent back through the error code parameter, it sets
C* the bytes provided field to the number of bytes it gives the
C* API for the parameter.
C*
C                     Z-ADD16        QUSBNB
C*
C* Initialize QCAPCMD options control block for CL processing
C*
C                     Z-ADD0         QCABCB
C                     MOVE '0'       QCABCC
C                     MOVE '0'       QCABCD
C                     MOVE '0'       QCABCF
C                     MOVE *BLANKS   QCABCG
C                     MOVE *LOVAL    QCABCH
C*
C* Create library QUEUELIB
C*
C                     MOVELCRTLIB    CMDSTR
C                     Z-ADD20        LENSTR
C*
C                     EXSR EXCCMD
C*
C* Create a data queue called EXAMPLEQ in library QUEUELIB.  The
C* queue will have a maximum entry length set at 10, and will be
C* FIFO (first-in first-out).
C*
C                     MOVELCRTDQ     CMDSTR
C                     Z-ADD42        LENSTR
C*
C                     EXSR EXCCMD
C*
C* Send information to the data queue.
C*
C                     CALL 'QSNDDTAQ'
C                     PARM 'EXAMPLEQ'QUENAM 10
C                     PARM 'QUEUELIB'LIBNAM 10
C                     PARM 10        MSGSZ   50
C                     PARM 'EXAMPLE' MSG    10
C*
C* Retrieve information from the data queue.
C*
C                     CALL 'QRCVDTAQ'
C                     PARM 'EXAMPLEQ'QUENAM 10
C                     PARM 'QUEUELIB'LIBNAM 10
C                     PARM 10        MSGSZ   50
C                     PARM           MSGBCK 10
```

```
C                     PARM 0         WAITTM  50
C*
C* Display the returned message
C*
C                     DSPLY          MSGBCK
C*
C* Delete the data queue
C*
C                     MOVELDLTDQ     CMDSTR
C                     Z-ADD31        LENSTR
C*
C                     EXSR EXCCMD
C*
C* Delete the library
C*
C                     MOVELDLTLIB    CMDSTR
C                     Z-ADD20        LENSTR
C*
C                     EXSR EXCCMD
C*
C                     SETON                     LR
C                     RETRN
C*
C* End of MAINLINE
C*
C*****************************************************************
C*
C           EXCCMD    BEGSR
C*
C* Process requested CL command
C*
C                     CALL 'QCAPCMD'
C                     PARM           CMDSTR
C                     PARM           LENSTR
C                     PARM           QCABC
C                     PARM           SIZE
C                     PARM 'CPOP0100'FORMAT  8
C                     PARM           RCVVAR  1
C                     PARM 0         RCVSIZ
C                     PARM           RCVSIZ
C                     PARM           QUSBN
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C           QUSBNC    IFGT 0
C                     MOVEL'QCAPCMD' APINAM
C                     EXSR ERRCOD
C                     ENDIF
C                     ENDSR
C*
C*****************************************************************
C*
C           ERRCOD    BEGSR
C*
C* Process errors returned from the API.
C*
C* If first error found, then open QPRINT *PRTF
C*
C           FSTERR    IFEQ '0'
C                     OPEN QPRINT
C                     MOVEL'1'       FSTERR
C                     ENDIF
C*
C* Print the error and the API that received the error
```

```
C*
C                     EXCPTBADNEW
C*
C                     ENDSR
OQPRINT  E  106           BADNEW
O                                    'Failed in API '
O                         APINAM
O                                    'with error '
O                         QUSBND
```

Related reference:

"Example in ILE C: Using data queues" on page 283
This ILE C program shows how to use APIs to create and manipulate a data queue.

Example in ILE RPG: Using data queues:

This ILE RPG program shows how to use APIs to create and manipulate a data queue.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
F*****************************************************************
F*****************************************************************
F*
F*  Program Name: DQUEUEX
F*
F*  Programming Language: ILE RPG
F*
F*  Description:  This program illustrates how to use APIs to
F*                create and manipulate a *DTAQ.
F*
F*  Header Files Included: QUSEC   - Error Code Parameter
F*                         QCAPCMD - Process Command API
F*
F*****************************************************************
F*
FQPRINT    O    F  132        PRINTER OFLIND(*INOF) USROPN
F*****************************************************************
D*
D* Error Code parameter include
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D* Process Command API Include
D*
D/COPY QSYSINC/QRPGLESRC,QCAPCMD
D*
D* Command strings
D*
D
DCRTLIB           C                   'CRTLIB LIB(QUEUELIB)'
DDLTLIB           C                   'DLTLIB LIB(QUEUELIB)'
DCRTDQ            C                   'CRTDTAQ DTAQ(QUEUELIB/+
D                                     EXAMPLEQ) MAXLEN(10)'
DDLTDQ            C                   'DLTDTAQ DTAQ(QUEUELIB/EXAMPLEQ)'
D*
D* Miscellaneous data structure
D*
DCMD_STR          S            100
DLEN_STR          S              9B 0
DCAP0100_SZ       S              9B 0 INZ(%SIZE(QCAP0100))
DRCVVAR_SZ        S              9B 0 INZ(0)
DAPI_NAME         S             10
DFIRST_ERR        S              1    INZ('0')
C*
C* Beginning of mainline
```

```
C*
C* Initialize the error code parameter.  To signal exceptions to
C* this program by the API, you need to set the bytes provided
C* field of the error code to zero.  Because this program has
C* exceptions sent back through the error code parameter, it sets
C* the bytes provided field to the number of bytes it gives the
C* API for the parameter.
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Initialize QCAPCMD options control block for CL processing
C*
C                   EVAL      QCACMDPT = 0
C                   EVAL      QCABCSDH = '0'
C                   EVAL      QCAPA = '0'
C                   EVAL      QCACMDSS = '0'
C                   EVAL      QCAMK = *BLANKS
C                   EVAL      QCAERVED = *LOVAL
C*
C* Create library QUEUELIB
C*
C                   EVAL      CMD_STR = CRTLIB
C                   EVAL      LEN_STR = %SIZE(CRTLIB)
C*
C                   EXSR      EXEC_CMD
C*
C* Create a data queue called EXAMPLEQ in library QUEUELIB.  The
C* queue will have a maximum entry length set at 10, and will be
C* FIFO (first-in first-out).
C*
C                   EVAL      CMD_STR = CRTDQ
C                   EVAL      LEN_STR = %SIZE(CRTDQ)
C*
C                   EXSR      EXEC_CMD
C*
C* Send information to the data queue.
C*
C                   CALL      'QSNDDTAQ'
C                   PARM      'EXAMPLEQ  '  NAME_OF_Q        10
C                   PARM      'QUEUELIB  '  NAME_OF_LB       10
C                   PARM      10            MSG_SZ            5 0
C                   PARM      'EXAMPLE   '  MSG              10
C*
C* Retrieve information from the data queue.
C*
C                   CALL      'QRCVDTAQ'
C                   PARM      'EXAMPLEQ  '  NAME_OF_Q
C                   PARM      'QUEUELIB  '  NAME_OF_LB
C                   PARM      10            MSG_SZ
C                   PARM                    MSG_BACK         10
C                   PARM      0             WAIT_TIME         5 0
C*
C* Display the returned message
C*
C                   DSPLY                   MSG_BACK
C*
C* Delete the data queue
C*
C                   EVAL      CMD_STR = DLTDQ
C                   EVAL      LEN_STR = %SIZE(DLTDQ)
C*
C                   EXSR      EXEC_CMD
C*
C* Delete the library
C*
C                   EVAL      CMD_STR = DLTLIB
C                   EVAL      LEN_STR = %SIZE(DLTLIB)
```

```
C*
C                   EXSR      EXEC_CMD
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C*
C* End of MAINLINE
C*
C*****************************************************************
C*
C     EXEC_CMD      BEGSR
C*
C* Process the requested CL command
C*
C                   CALL      'QCAPCMD'
C                   PARM                    CMD_STR
C                   PARM                    LEN_STR
C                   PARM                    QCAP0100
C                   PARM                    CAP0100_SZ
C                   PARM      'CPOP0100'    FORMAT            8
C                   PARM                    RCVVAR            1
C                   PARM      0             RCVVAR_SZ
C                   PARM                    RCVVAR_SZ
C                   PARM                    QUSEC
C*
C* If an exception occurs, the API returns the exception in the
C* error code parameter.  The bytes available field is set to
C* zero if no exception occurs and greater than zero if an
C* exception does occur.
C*
C                   IF        QUSBAVL > 0
C                   EVAL      API_NAME = 'QCAPCMD'
C                   EXSR      ERRCOD
C                   ENDIF
C                   ENDSR
C*
C*****************************************************************
C*
C     ERRCOD        BEGSR
C*
C* Process errors returned from the API.
C*
C* If first error found, then open QPRINT *PRTF
C*
C                   IF        FIRST_ERR = '0'
C                   OPEN      QPRINT
C                   EVAL      FIRST_ERR = '1'
C                   ENDIF
C*
C* Print the error and the API that received the error
C*
C                   EXCEPT    BAD_NEWS
C*
C                   ENDSR
OQPRINT    E            BAD_NEWS       1
O                                              'Failed in API '
O                       API_NAME
O                                              'with error '
O                       QUSEI
```

Related reference:

"Example in ILE C: Using data queues" on page 283
This ILE C program shows how to use APIs to create and manipulate a data queue.

Example in ILE C: Using user queues:

This ILE C program shows how to use APIs to create and manipulate a user queue.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*                                                                   */
/*Program Name: UQUEUEX                                              */
/*                                                                   */
/*Program Language:  ILE C                                           */
/*                                                                   */
/*Description:  This program illustrates how to use APIs to create   */
/*              and manipulate a user queue.                         */
/*                                                                   */
/*                                                                   */
/*Header Files Included: <stdio.h>                                   */
/*                       <signal.h>                                  */
/*                       <string.h>                                  */
/*                       <stdlib.h>                                  */
/*                       <miptrnam.h>                                */
/*                       <miqueue.h>                                 */
/*                       <pointer.h>                                 */
/*                       <quscrtuq.h>                                */
/*                       <qusdltuq.h>                                */
/*                       <qusec.h>                                   */
/*                                                                   */
/*APIs Used:     QUSCRTUQ - Create a user queue                      */
/*               QUSDLTUQ - Delete a user queue                      */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/*********************************************************************/
/*                           Includes                                */
/*********************************************************************/

#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <milib.h>                      /* from QCLE/h               */
#include <miptrnam.h>                   /* from QCLE/h               */
#include <miqueue.h>                    /* from QCLE/h               */
#include <pointer.h>
#include <quscrtuq.h>                   /* from QSYSINC/h            */
#include <qusdltuq.h>                   /* from QSYSINC/h            */
#include <qusec.h>                      /* from QSYSINC/h            */

/*********************************************************************/
/*                           Structures                              */
/*********************************************************************/

typedef struct {
 Qus_EC_t ec_fields;
 char     exception_data[100];
} error_code_struct;

/*********************************************************************/
/*                                                                   */
/*                              Main                                 */
/*                                                                   */
/*********************************************************************/

void main()
{
  char text_desc[50];
```

```
error_code_struct error_code;
_SYSPTR queuelib_sysptr,
        user_queue_obj_sysptr;
_RSLV_Template_T  rslvsp_template;
_ENQ_Msg_Prefix_T enq_msg_prefix;
_DEQ_Msg_Prefix_T deq_msg_prefix;
char enq_msg[50],
     deq_msg[50];
int success=0;

/*********************************************************************/
/* Create a library to create the user queue into.                   */
/*********************************************************************/

system("CRTLIB LIB(QUEUELIB)");

/*********************************************************************/
/* Initialize the error code parameter.                              */
/*********************************************************************/
error_code.ec_fields.Bytes_Provided=sizeof(error_code_struct);

/*********************************************************************/
/* Call the QUSCRTUQ API to create a user queue.                     */
/*                                                                   */
/*   This will create a user queue called EXAMPLEQ in library        */
/*   QUEUELIB, with the following attributes:                        */
/*                                                                   */
/*     1.  Extended attribute of "VALID     ", which could have      */
/*         been any valid *NAME.                                     */
/*     2.  A queue type of "F", or First-in, first-out.              */
/*     3.  A key length of 0.  If the queue is not keyed, this       */
/*         value must be 0.                                          */
/*     4.  A maximum message size of 10 bytes.  This number can      */
/*         be as large as 64K bytes.                                 */
/*     5.  The initial number of messages set to 10.                 */
/*     6.  Additional number of messages set to 10.                  */
/*     7.  Public authority of *USE.                                 */
/*     8.  A valid text description.                                 */
/*     9.  Replace option of *YES. This means that if a user queue   */
/*         already exists by the name specified, in the library      */
/*         specified, that it will be replaced by this               */
/*         request.                                                  */
/*    10.  Domain value of *USER.                                    */
/*    11.  Pointer value of *NO.  Messages in the queue cannot       */
/*         contain pointer data.                                     */
/*********************************************************************/

memcpy(text_desc, "THIS IS TEXT FOR THE EXAMPLE USER QUEUE          ",
       50);

QUSCRTUQ("EXAMPLEQ  QUEUELIB  ",  /* Qualified user queue name     */
         "VALID     ",           /* Extended attribute            */
         "F",                    /* Queue type                    */
         0,                      /* Key length                    */
         10,                     /* Maximum message size          */
         10,                     /* Initial number of messages    */
         10,                     /* Additional number of messages */
         "*ALL      ",           /* Public authority              */
         text_desc,              /* Text Description              */
         "*YES      ",           /* Replace existing user queue   */
         &error_code,            /* Error code                    */
         "*USER     ",           /* Domain of user queue          */
         "*NO       ");          /* Allow pointer data            */

/*********************************************************************/
/* If an exception occurred, the API would have returned the         */
/* exception in the error code parameter.  The bytes available       */
```

```
/* field will be set to zero if no exception occurred and greater  */
/* than zero if an exception did occur.                            */
/*******************************************************************/

if (error_code.ec_fields.Bytes_Available > 0)
{
   printf("ATTEMPT TO CREATE A USER QUEUE FAILED WITH EXCEPTION:%.7s",
          error_code.ec_fields.Exception_Id);
   exit(1);
}

/*******************************************************************/
/* Send information to the queue.                                  */
/*                                                                 */
/* We will need to use MI instructions to accomplish this.         */
/* There are three steps that must be done:                        */
/*                                                                 */
/* 1. Resolve a system pointer to the library containing the user  */
/*    queue object.                                                */
/* 2. Using the system pointer to the library, resolve a system    */
/*    pointer to user queue object in the library.                 */
/* 3. Enqueue the entry using the system pointer for the user      */
/*    queue.                                                       */
/*                                                                 */
/*******************************************************************/

/*******************************************************************/
/* First we must resolve to library QUEUELIB.                      */
/*******************************************************************/
memset(rslvsp_template.Obj.Name,' ',30);
memcpy(rslvsp_template.Obj.Name,"QUEUELIB",8);
rslvsp_template.Obj.Type_Subtype = _Library; /* found in milib.h   */
rslvsp_template.Auth = _AUTH_NONE; /* found in milib.h             */

_RSLVSP6(&queuelib_sysptr,         /* system pointer to be set     */
         &rslvsp_template,         /* resolve template             */
         &rslvsp_template.Auth);   /* authority to set in sysptr   */

/*******************************************************************/
/* We can now resolve to the user queue object.  We will pass the  */
/* system pointer to library QUEUELIB to RSLVSP so the resolve     */
/* will only search library QUEUELIB for the user queue object.    */
/* This is necessary so that we ensure that we are using the       */
/* correct object.                                                 */
/*******************************************************************/
memset(rslvsp_template.Obj.Name,' ',30);
memcpy(rslvsp_template.Obj.Name, "EXAMPLEQ", 8);
rslvsp_template.Obj.Type_Subtype = _Usrq; /* found in milib.h      */
rslvsp_template.Auth = _AUTH_ALL;  /* found in milib.h             */

_RSLVSP8(&user_queue_obj_sysptr,   /* system pointer to be set     */
         &rslvsp_template,         /* resolve template             */
         &queuelib_sysptr,         /* sysptr to library            */
         &rslvsp_template.Auth);   /* authority to set in sysptr   */

/*******************************************************************/
/* Enqueue the entry.                                              */
/*******************************************************************/
enq_msg_prefix.Msg_Len = 10;
enq_msg_prefix.Msg[0] = '\0';        /* Only used for keyed queues*/
memcpy(enq_msg, "EXAMPLE   ", 10);

_ENQ(&user_queue_obj_sysptr,       /* system pointer to user queue */
     &enq_msg_prefix,              /* message prefix               */
     (_SPCPTR)enq_msg);            /* message text                 */

/*******************************************************************/
```

```
  /* Dequeue the entry.                                               */
  /*******************************************************************/
  success = _DEQI(&deq_msg_prefix,   /* message prefix               */
                  (_SPCPTR)deq_msg,  /* message text                 */
                  &user_queue_obj_sysptr); /* sys ptr to user queue  */

  if(success)
  {
    printf("Queue entry information: %.10s\n", deq_msg);
  }
  else
  {
    printf("Entry not dequeued\n");
  }

  /*******************************************************************/
  /* Delete the user queue.                                          */
  /*******************************************************************/

  QUSDLTUQ("EXAMPLEQ  QUEUELIB  ",  /* Qualified user queue name     */
           &error_code);            /* Error code                    */

  /*******************************************************************/
  /* If an exception occurred, the API would have returned the       */
  /* exception in the error code parameter.  The bytes available     */
  /* field will be set to zero if no exception occurred and greater  */
  /* than zero if an exception did occur.                            */
  /*******************************************************************/

  if (error_code.ec_fields.Bytes_Available > 0)
  {
     printf("ATTEMPT TO DELETE A USER QUEUE FAILED WITH EXCEPTION:%.7s",
            error_code.ec_fields.Exception_Id);
     exit(1);
  }

  /*******************************************************************/
  /* Delete the library created for this example.                    */
  /*******************************************************************/

  system("DLTLIB LIB(QUEUELIB)");
}
```

Examples: APIs and exit programs

These examples show how to use a wide variety of APIs and exit programs.

Note: To use these examples, you need the header files in the system include (QSYSINC) library.

Related concepts:

"Include files and the QSYSINC library" on page 59
An *Include file* is a text file that contains declarations that are used by a group of functions, programs, or users. The system include (QSYSINC) library provides all source include files for APIs that are included with the IBM i operating system.

"Performing tasks using APIs" on page 238
You can use APIs to perform different types of tasks.

Example: 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 have the following options:

- 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:

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 /* ************************************************************ */
 /* 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(&ELENB)
            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.

Example: Changing a job schedule entry

This command interface to the Change Job Schedule Entry User (CHGSCDEUSR) program can change the user for a job schedule entry.

You have the following options:

- 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:

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/* ************************************************************ */
/* 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 *NE ' ') THEN(GOTO +
                         CMDLBL(ENDLOOP))

            IF         COND(&USERNM *NE &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.

Example: Creating a batch machine

These ILE C programs emulate a batch machine. One program acts as a requester and puts the entries into a user space. The other program acts as a server, takes the entries from the user queue, and runs the request.

The following APIs are used in this example:

- Create User Queue (QUSCRTUQ)
- Execute Command (QCMDEXC)

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Requester program ($USQEXREQ)

```
/*********************************************************************/
/*  PROGRAM:  $USQEXREQ                                              */
/*                                                                   */
/*  LANGUAGE:  ILE C                                                 */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM ENTERS COMMANDS TO BE PROCESSED ONTO */
/*    A QUEUE CALLED 'TESTQ' IN LIBRARY 'QGPL'.  THE USER WILL BE   */
/*    PROMPTED TO ENTER AS MANY COMMANDS (UNDER 51 CHARACTERS) AS   */
/*    IS DESIRED.  WHEN THE USER WISHES TO END THE PROGRAMS,        */
/*    ALL THAT NEED BE DONE IS ENTER 'quit' AT THE PROMPT.          */
/*                                                                   */
/*  APIs USED:                                                       */
/*                                                                   */
/*********************************************************************/
#include <stdio.h>
#include <string.h>
#include <miqueue.h>
#include <miptrnam.h>


main()
{
_ENQ_Msg_Prefix_T e_msg_prefix;
_SYSPTR queue;
char INMsg[100];

/*********************************************************************/
/*  Resolve to the queue created by $USQEXSRV.                       */
/*********************************************************************/

queue = rslvsp(_Usrq,"TESTQ","QGPL",_AUTH_ALL);
e_msg_prefix.Msg_Len = 100;

/*********************************************************************/
/*  Loop until the user enters 'quit' as the command.                */
/*********************************************************************/

  while (1) {
    printf("\nEnter command to put on queue, or 'quit' \n ");
    scanf("%100s", INMsg);
    gets(INMsg);
    printf("\nCommand entered was ==> %.100s\n",INMsg);

/*********************************************************************/
/*  Check to see if the user entered 'quit' as the command.          */
/*  If true then break out of the 'while' loop.                      */
/*********************************************************************/

    if ((strncmp(INMsg,"quit",4) == 0)||(strncmp(INMsg,"QUIT",4) == 0))
      { break; }

/*********************************************************************/
/*  Add the user-entered command to the queue.                       */
/*********************************************************************/

    enq(queue,&e_msg_prefix,INMsg);
    strcpy(INMsg," ");
  } /*while*/

/*********************************************************************/
/*  Add the command end to the queue which causes the                */
/*  server program ($USQEXSRV) to end                                */
```

```
/*******************************************************************/

  strcpy(INMsg,"END");
  enq(queue,&e_msg_prefix,INMsg);
} /* $USQEXREQ */
```

To create the requester program using ILE C, specify the following:

```
CRTBNDC PGM(QGPL/$USQEXREQ) SRCFILE(QGPL/QCSRC)
```

Server program ($USQEXSRV)

```
/*******************************************************************/
/*  PROGRAM:  $USQEXSRV                                            */
/*                                                                 */
/*  LANGUAGE:  ILE C                                               */
/*                                                                 */
/*  DESCRIPTION:  THIS PROGRAM EXTRACTS COMMANDS TO BE RUN FROM    */
/*    A QUEUE CALLED 'TESTQ' IN LIBRARY 'QGPL'.  THE COMMANDS WILL */
/*    BE EXTRACTED AND RUN IN FIFO ORDER.  THE QUEUE WILL BE       */
/*    CREATED PRIOR TO USE AND SHOULD BE DELETED AFTER EACH USE    */
/*    OF THIS EXAMPLE PROGRAM.  THIS PROGRAM END WHEN IT           */
/*    EXTRACTS THE COMMAND 'END' FROM THE QUEUE.                   */
/*       THE FLOW IS AS FOLLOWS:                                   */
/*         (1)  CREATE THE USER QUEUE                              */
/*         (2)  ENTER LOOP                                         */
/*         (3)     WAIT FOREVER FOR A COMMAND ON THE QUEUE         */
/*         (4)      IF COMMAND IS 'END' THEN EXIT LOOP             */
/*         (5)      ELSE RUN COMMAND, RESTART LOOP                 */
/*         (6)  END LOOP                                           */
/*    FOR BEST RESULTS, THIS PROGRAM CAN BE CALLED BY THE USER, THEN*/
/*    THE $USQEXREQ SHOULD BE CALLED FROM ANOTHER SESSION.         */
/*                                                                 */
/* APIs USED:  QCMDEXC, QUSCRTUQ                                   */
/*                                                                 */
/*******************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <micomput.h>
#include <miqueue.h>
#include <miptrnam.h>
#include <quscrtuq.h>
#include <qcmdexc.h>


main()
{
_DEQ_Msg_Prefix_T d_msg_prefix;
_SYSPTR           queue;
char              OUTMsg[100];
int               cmd_name_lngth;
decimal(15,5)     pack_name_lngth;
char              igc_param[] = "IGC";

/*******************************************************************/
/*    Set up the parameters to be used in the call to 'QUSCRTUQ'   */
/*******************************************************************/

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 .....    ";
```

```
/*****************************************************************/
/*  Call the 'QUSCRTUQ' program to create the user queue.        */
/*****************************************************************/

  QUSCRTUQ(q_name,ext_atr,q_type,key_lngth,max_msg_s,int_msgs,
           add_msgs,auth,desc);

/*****************************************************************/
/*  Resolve to the queue created above.                          */
/*****************************************************************/

  queue = rslvsp(_Usrq,"TESTQ","QGPL",_AUTH_ALL);

/*****************************************************************/
/*  Set the deq operation to wait for command indefinitely.      */
/*****************************************************************/

  d_msg_prefix.Wait_Forever = 1;

/*****************************************************************/
/*  Loop until the command 'END' is extracted from the queue     */
/*****************************************************************/

  while (1) {
    deq(&d_msg_prefix,OUTMsg,queue);

/*****************************************************************/
/*  Check to see if the command extracted is 'END'               */
/*  If true then break out of the 'while' loop.                  */
/*****************************************************************/

    if (strncmp(OUTMsg,"END",3) == 0)
      { break; }
    cmd_name_lngth = strlen(OUTMsg);

/*****************************************************************/
/*  Convert the integer in cmd_name_lngth to a packed decimal    */
/*****************************************************************/

    cpynv(NUM_DESCR(_T_PACKED,15,5), &pack_name_lngth,
          NUM_DESCR(_T_SIGNED,4,0), &cmd_name_lngth);

/*****************************************************************/
/*  Execute the command extracted from the queue                 */
/*****************************************************************/

    QCMDEXC(OUTMsg,pack_name_lngth,igc_param);
  } /* while */
}  /* $USQEXSRV */
```

To create the server program using ILE C, specify the following:

```
CRTBNDC PGM(QGPL/$USQEXSRV) SRCFILE(QGPL/QCSRC)
```

Example: Creating and manipulating a user index

This example shows how to create and manipulate a user index with a call from an MI program.

For another example using the Create User Index (QUSCRTUI) API, see “Example: Creating your own telephone directory” on page 314.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 571.

```
/*****************************************************************/
/*                                                               */
/* 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;           /* OCCURRENCE 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;
```

Example: Creating your own telephone directory

These ILE C programs show how to create a user index for a telephone directory, how to insert entries into the telephone directory, and how to find an entry.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

To create a user index, use the following program, $USIDXCRT:

```
/*********************************************************************/
/*  PROGRAM:  $USIDXCRT                                              */
/*                                                                   */
/*  LANGUAGE:  ILE C                                                 */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM CREATES A USER INDEX NAMED "TESTIDX"  */
/*   IN THE LIBRARY "QGPL".                                          */
/*                                                                   */
/*  APIs USED:  QUSCRTUI                                             */
/*                                                                   */
/*********************************************************************/
#include <quscrtui.h>
main()
{
/*********************************************************************/
/*    Set up the parameters to be used in the call to 'QUSCRTUI'     */
/*********************************************************************/
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 .....    ";
/*********************************************************************/
/*  Call the 'QUSCRTUI' program 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

```
CRTBNDC PGM(QGPL/$USIDXCRT) SRCFILE(QGPL/QCSRC)
```

.

To insert entries into the user index, use the following program $USIDXEX:

```
/*********************************************************************/
/*  PROGRAM:  $USIDXEX                                               */
/*                                                                   */
/*  LANGUAGE:  ILE C                                                 */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM USES A USER INDEX TO KEEP TRACK OF    */
/*    NAMES AND PHONE NUMBERS.  THERE ARE TWO OPERATIONS THAT ARE    */
/*    DEMONSTRATED IN THIS EXAMPLE.  THE FIRST IS THE INSERTION OF   */
/*    AN ENTRY INTO THE INDEX, AND SECONDLY THE FINDING OF A GIVEN   */
/*    INDEX ENTRY.                                                   */
/*    THE INDEX IS KEYED ON THE LAST NAME, THEREFORE ENTER AS MUCH   */
/*    OF THE NAME AS YOU KNOW AND THE PROGRAM WILL LIST ALL ENTRIES */
/*    MATCHING YOUR STRING (IN ALPHABETICAL ORDER).                  */
/*                                                                   */
/*  APIs USED:  NONE                                                 */
```

```
/*                                                                  */
/*******************************************************************/
#include <stdio.h>
#include <string.h>
#include <miindex.h>
#include <miptrnam.h>
#include <stdlib.h>
#include <ctype.h>

_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;


/*******************************************************************/
/* Procedure to copy 'cpylngth' elements of 'string2' into the     */
/* new string, 'string1';  starting at position 'strpos'.          */
/*******************************************************************/

void strncpyn(string1,string2,strpos,cpylngth)
 char string1[],string2[];
 int strpos,cpylngth;
{
 int x = 0;
 while (x < cpylngth)
  string1[x++]=string2[strpos++];
} /*strncpyn*/


/*******************************************************************/
/* Procedure to convert any string into uppercase, where applicable */
/*******************************************************************/

void convert_case(string1)
 char string1[];
{
 int x = 0;
 while (x < (strlen(string1))) {
  string1[x] = toupper(string1[x]);
  x++;
 } /*while*/
} /*convert_case*/


main()
{
fnd_option_list = malloc(sizeof(_IIX_Opt_List_T)
                       +99*sizeof(_IIX_Entry_T));

/*******************************************************************/
/*  Resolve to the index created in $USIDXCRT.                     */
/*******************************************************************/

  index = rslvsp(_Usridx,"TESTIDX","QGPL",_AUTH_ALL);

/*******************************************************************/
/*  Set up the insert option list                                  */
/*******************************************************************/

  ins_option_list.Rule = _INSERT_REPLACE;
```

```
  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;

/*********************************************************************/
/*  Set up the find option list                                      */
/*********************************************************************/

  fnd_option_list->Rule = _FIND_EQUALS;
  fnd_option_list->Occ_Count = 100;

/*********************************************************************/
/*  Loop 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,Integrated Netfinity Server_option_list);
   } /* if 'a'*/
  if ((strncmp(response,"L",1)==0)||(strncmp(response,"l",1)==0))
   {
     printf("\nEnter name to find. ex(Last, First)\n");
     gets(In_Name);
     convert_case(In_Name);
     fnd_option_list->Arg_Length = strlen(In_Name);
     fndinxen(Out_Num,index,fnd_option_list,In_Name);
     length_of_entry = fnd_option_list->Entry[0].Entry_Length;
     Ent_Found = fnd_option_list->Ret_Count;
     if (Ent_Found == 0)
       printf("\nName not found in index => %s\n",In_Name);
     else {
       if (Ent_Found > 1)  {
        printf("\n%d occurences found,\n",Ent_Found);
        count = 0;
        start = 0;
        while (count++ < Ent_Found) {
         printf("Name and number is => %s\n",Out_Num);
         start = start + length_of_entry;
         strncpyn(Out_Num,Out_Num,start,length_of_entry);
        } /* while */
       }else
        printf("\nName and number is => %s\n",Out_Num);
     }  /*else*/
   }  /*if 'l'*/
  if ((strncmp(response,"Q",1)==0)||(strncmp(response,"q",1)==0))
   { break; }
 }  /*while*/
} /*$USIDXEX*/
```

To create the ILE C program to insert entries into the user index, specify

```
CRTBNDC PGM(QGPL/$USIDXEX) SRCFILE(QGPL/QCSRC)
```

.

Related reference:

"Example: Creating and manipulating a user index" on page 309
This example shows how to create and manipulate a user index with a call from an MI program.

Example: Defining queries

These ILE C programs show how to use the Query (QQQQRY) API to define a simple query for ordering; a join query; and a join query with selection, grouping, and ordering.

These programs use the QQAPI header (or include) file and the QQFUNCS query code.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

QQAPI header

```
/*********************************************************************/

#ifndef _QQAPIH
#define _QQAPIH


/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  Defines constants and structures for use               */
/*            with the QQQQRY API examples.                          */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: None                                                   */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/


/* The following define will enable some debug procedures and code */
/* #define QQDEBUG */

/* Query Open options */
#define QO_INPUT   1
#define QO_OUTPUT  2
#define QO_UPDATE  4
#define QO_DELETE  8

/* simple defines */
#define ON    1
#define OFF   0

/* user defined limits - change as needed */
#define MAX_ORDERBY    20
  /* max number of order by fields (8000 max)*/
#define MAX_JOINTESTS  20
  /* max number of join tests (999 max)*/
#define MAX_GROUPBY    20
  /* max number of order by fields (120 max)*/
/* storage sizes - increase if needed */
#define QDT_SIZE      6000
#define FORMAT_SIZE   5000
#define SELECT_SIZE   5000
#define AP_SIZE      65535     /* Initialize access plan size to 64K */
```

```
/* Required definitions - do NOT change, hard limits */
#define MAX_FILES      32      /* Maximum number of files in a query */
#define REQ_REL       "01"     /* Required value for release field */
#define REQ_VER       "00"     /* Required value for version field */
#define QFLD_SIZE      30      /* QQ API field size - see qqqqry.h */

/* define error code structure */
typedef struct
{
    int bytes_provided;
    int bytes_available;
    char msgid[(7]);
    char reserved;
    char data[(512]);
} error_code;

/* define attribute record for QUSCUSAT API */
typedef _Packed struct
{
    int  numAttrs;
    int  key;
    int  length;
    _Packed union {
      long spaceSize;    /* key = 1 */
      char initialValue; /* key = 2 */
      char autoExtend;   /* key = 3 */
    } data;
} QUSCUSAT_T;

/* define access plan structure */
typedef _Packed struct
{
    _SPCPTR storagePtr;
    long size;
    char reserved[(28]);
} ACCPLN_T;


/* Function prototypes: */
void dumpPtr(char *, char *, int );
char *strcnv400(char *, int );
int strcpy400(char *, char *, int );
void initUFCB(QDBUFCB_T *, int , Qdb_Qddfmt_t *);
void initQDT(QDBQH_T *, char , int , int ,
  char , int );
void initFile(QDBQFHDR_T *, char , char );
void initFormat(Qdb_Qddfmt_t *, char *);
void initSelection(QDBQS_T *);
void initOrderBy(QDBQKH_T *);
void initGroupBy(QDBQGH_T *);
void initJoin(QDBQJHDR_T *);
int addFile(QDBQFHDR_T *, QDBQN_T *,
  char *, char *, char *, char *);
int getRecordFmt(Qdb_Qddfmt_t *, long,
  char *, char *, char *);
long copyField(Qdb_Qddfmt_t *, char *, int ,
  Qdb_Qddfmt_t *);
void setFieldUsage(Qdb_Qddfmt_t *, char *, char );
int addSelectField(QDBQS_T *, char *, int );
int addSelectLiteral(QDBQS_T *, void *, int );
int addSelectOperator(QDBQS_T *, char *);
int addOrderBy(QDBQKH_T *, QDBQKF_T *,
  char *, int );
int addGroupBy(QDBQGH_T *, QDBQGF_T *,
  char *, int );
```

```
int addJoinTest(QDBQJHDR_T *, QDBQJFLD_T *, char *,
  int , char *, int , char *);
void addQDTsection(QDBQH_T *, char *, int , int *);
long createAccessPlanSpace(ACCPLN_T *, char *, long );
int saveQDT(QDBQH_T *, ACCPLN_T *);
int saveAccessPlan(ACCPLN_T *);
int loadQDT(QDBQH_T *, ACCPLN_T *);
long loadAccessPlan(ACCPLN_T *, char *);

#endif
/*********************************************************************/
```

QQFUNCS query code

```
/*********************************************************************/
#include <stdio.h>
#include <string.h>
#include <qdbrtvfd.h>
#include <qqqqry.h>
#include <quscrtus.h>
#include <qusptrus.h>
#include <quscusat.h>
#include <qusrusat.h>
#include "qqapi.h"

/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  This module contains all of the functions              */
/*            used by the examples to build the API information.     */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: QDBRTVFD, QUSCRTUS, QUSCUSAT, QUSPTRUS, QUSRUSAT       */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#ifdef QQDEBUG
/* dumpPtr(comment string, pointer, length)
 - prints a comment then dumps data in hexadecimal starting at the
   given pointer location for the specified length */
void dumpPtr(char *text, char *ptr, int len)
{
    int i;

    printf("%s\n", text);
    for (i=0; i < len; i++, ptr++)
    {
        printf("%02X ", (int) *ptr);
        if ((i+1) % 16 == 0)
           printf("\n");
    }
    printf("\n");
}
#endif


/* strcnv400(source string, string length)
 - convert a string to a zero terminated string */
char *strcnv400(char *str, int len)
{
static char buffer[256];

    strncpy(buffer, str, len);
    buffer[len] = (char) 0;
    return(buffer);
```

```
}


/* strcpy400(destination string, source string, source length)
 - copy a zero terminated string to a string, pad with blanks
   if necessary */
int strcpy400(char *dest, char *src, int len)
{
    int i;

    if ((i = strlen(src)) > len)
        len = i;
    if (len)
        memcpy(dest, src, strlen(src));
    if (i < len)
        memset((dest+i), ' ', len-i);
    return(len);
}


/* initUFCB(ufcb, open options, record format)
 - initialize the UFCB structure */
void initUFCB(QDBUFCB_T *ufcbPtr, int openFlags,
  Qdb_Qddfmt_t *formatPtr)
{
    _Packed struct qufcb *ufcb;

    /* verify parameters */
    if (ufcbPtr == NULL || openFlags == 0)
    {
        printf("Invalid UFCB settings\n");
        return;
    }
    /* Clear the entire UFCB */
    memset((void *) ufcbPtr, (char) 0, sizeof(QDBUFCB_T));
    /* Now start initializing values */
    ufcb = &ufcbPtr-&gt;qufcb;
    strcpy400((char *) ufcb->relver.release, REQ_REL,
      sizeof(ufcb->relver.release));
    strcpy400((char *) ufcb->relver.version, REQ_VER,
      sizeof(ufcb->relver.version));
    /* Blocked Records (BLKRCD) should be on if CPYFRMQRYF is used */
    ufcb->markcnt.flg2brcd = ON;
    ufcb->parameter.maximum = MAXFORMATS;
    /* Set the open option */
    if (openFlags&QO_INPUT)
        ufcb->open.flagui = ON;
    if (openFlags&QO_OUTPUT)
        ufcb->open.flaguo = ON;
    if (openFlags&QO_UPDATE)
        ufcb->open.flaguu = ON;
    if (openFlags&QO_DELETE)
        ufcb->open.flagud = ON;
    /* set up options to match _Ropen options */
    ufcb->parameter.keyfdbk = KEYFDBK;
    ufcb->parameter.keyonoff = ON; /* Key feedback ON */
    ufcb->parameter.filedep = FILEDEP;
    ufcb->parameter.fldonoff = ON; /* File dependent I/O ON */
    /* turn the rest of the parameters off */
    ufcb->parameter.seqonly = NOTSEQUPROC;
    ufcb->parameter.primrln1 = NOTRECORDLTH;
    ufcb->parameter.commitc = NOTCOMITCTL;
    /* if the format is supplied,
      define it in the UFCB and do level checking */
    if (formatPtr != NULL)
    {
        ufcb->parameter.lvlchk = LEVELCK;
```

```
        ufcb->parameter.lvlonoff = ON; /* Level check ON */
        ufcb->parameter.curnum = 1; /* only one format */
        /* set the format name and format level identifier */
        ufcb->parameter.recfmts = FORMATSEQ;
        memcpy(ufcb->parameter.formats[0].name, formatPtr->Qddfname,
          sizeof(ufcb->parameter.formats[0].name));
        memcpy(ufcb->parameter.formats[0].number, formatPtr->Qddfseq,
          sizeof(ufcb->parameter.formats[0].number));
    }
    else /* no format and level checking */
    {
        ufcb->parameter.lvlchk = NOTLEVELCK;
        ufcb->parameter.recfmts = NOTFORMATSEQ;
    }
    ufcb->ufcbend  = ENDLIST;
}


/* initQDT(qdt, options...)
 - initialize the QDT header */
void initQDT(QDBQH_T *qdtHdr, char alwCpyDta,
  int optAllAp, int statusMsgs,
  char optimize, int forRows)
{
    if (qdtHdr == NULL)
    {
        printf("Invalid QDT settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the entire QDT */
    memset((void *) qdtHdr, (char) 0, sizeof(QDBQH_T));
    /* set the initial QDT space used size */
    qdtHdr->qdbspcsize = sizeof(QDBQH_T);
    /* QDT options... */
    /* ordering not specified */
    qdtHdr->qdbqkeyo = -1;
    /* set optimize parameter (ALLIO, FIRSTIO, MINWAIT) */
    if (optimize == QDBQFINA || optimize == QDBQFINF ||
      optimize == QDBQFINM || optimize == QDBQFINC)
        qdtHdr->qdbqfin = optimize; /* OPTIMIZE() parameter */
    else
        qdtHdr->qdbqfin = QDBQFINA; /* default to OPTIMIZE(*ALLIO) */
    /* set allow copy data parameter (YES, NO, OPTIMIZE) */
    if (alwCpyDta == QDBQTEMN || alwCpyDta ==  QDBQTEMO ||
      alwCpyDta == QDBQTEMA)
        qdtHdr->qdbqtem = alwCpyDta; /* ALWCPYDTA() parameter */
    else
        qdtHdr->qdbqtem = QDBQTEMA; /* default to ALWCPYDTA(*YES) */
    /* status messages (YES, NO) */
    qdtHdr->qdbqattr.qdbqnst = statusMsgs ? ON : OFF;
    /* optimize all access path parameter (YES, NO) */
    qdtHdr->qdbqdt_7.qdbqopta = optAllAp ? ON : OFF;
    /* optimizer for n rows parameter */
    qdtHdr->qdbq_optmrows = forRows > 0 ? forRows : 0;
}


/* initFile(file section, join type, join order option)
 - initialize the file header section */
void initFile(QDBQFHDR_T *fileHdr, char joinType, char joinOrder)
{
    if (fileHdr == NULL)
    {
        printf("Invalid File Header settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
```

```
    memset((void *) fileHdr, (char) 0, sizeof(QDBQFHDR_T));
    /* File Spec options... */
    /* inner, partial outer or exception join type */
    if (joinType == QDBQINNJ || joinType == QDBQOUTJ ||
      joinType == QDBQEXCJ)
        fileHdr->qdbqmfop = joinType;
    else
        fileHdr->qdbqmfop = QDBQINNJ;
    /* join order - any order or join as specified */
    fileHdr->qdbqmfor = joinOrder == QDBQMFON ? QDBQMFON : QDBQMFOA;
}


/* initFormat(format section, format name)
 - initialize the format header section */
void initFormat(Qdb_Qddfmt_t *formatHdr, char *name)
{
    if (formatHdr == NULL)
    {
        printf("Invalid Format Header settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) formatHdr, (char) 0, sizeof(Qdb_Qddfmt_t));
    /* Format Spec options... */
    strcpy400(formatHdr->Qddfname, name, sizeof(formatHdr->Qddfname));
    formatHdr->Qddfrcid = 65535;
    formatHdr->Qddfsrcd = 65535;
    formatHdr->Qddflgs.Qddfrsid = 1;
    memset(formatHdr->Qddfseq, ' ', sizeof(formatHdr->Qddfseq));
    memset(formatHdr->Qddftext, ' ', sizeof(formatHdr->Qddftext));
    /* Format size (so far) */
    formatHdr->Qddbyava = sizeof(Qdb_Qddfmt_t);
    formatHdr->Qddbyrtn = formatHdr->Qddbyava;
}


/* initSelection(selection section)
 - initialize the selection header section */
void initSelection(QDBQS_T *selectHdr)
{
    if (selectHdr == NULL)
    {
        printf("Invalid selection settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) selectHdr, (char) 0, sizeof(QDBQS_T));
    /* set initial selection spec size (minus dummy selection spec) */
    selectHdr->qdbqsl = sizeof(QDBQS_T) - sizeof(selectHdr->qdbqspec);
}


/* initOrderBy(orderby section)
 - initialize order by header section */
void initOrderBy(QDBQKH_T *orderByHdr)
{
    if (orderByHdr == NULL)
    {
        printf("Invalid Order By settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) orderByHdr, (char) 0, sizeof(QDBQKH_T));
}
```

```
/* initGroupBy(groupby section)
 - initialize group by header section */
void initGroupBy(QDBQGH_T *groupByHdr)
{
    if (groupByHdr == NULL)
    {
        printf("Invalid Group By settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) groupByHdr, (char) 0, sizeof(QDBQGH_T));
}


/* initJoin(join section)
 - initialize join header section */
void initJoin(QDBQJHDR_T *joinHdr)
{
    if (joinHdr == NULL)
    {
        printf("Invalid Join settings\n");
        return;  /* invalid pointer */
    }
    /* Clear the header */
    memset((void *) joinHdr, (char) 0, sizeof(QDBQKH_T));
    /* set initial join spec size */
    joinHdr->qdbqjln = sizeof(QDBQJHDR_T);
}


/* addFile (file section, file spec section, file name, file library,
   file member, file format)
 - add file information to the file section */
int addFile(QDBQFHDR_T *fileHdr, QDBQN_T *fileSpec,
  char *filename, char *library, char *member, char *format)
{
    int i;
    QDBQFLMF_T *fileSpecPtr;

    if (fileHdr == NULL || fileSpec == NULL || filename == NULL)
        return(0);  /* invalid data */
    if (fileHdr->qdbqfilnum == MAX_FILES)
        return(0);  /* no more files allowed */
    /* increment the count of file specs */
    i = fileHdr->qdbqfilnum++;
    /* initialize the file spec area */
    memset((void *) &fileSpec[i], (char) 0, sizeof(QDBQN_T));
    fileSpecPtr = (QDBQFLMF_T *) &fileSpec[i].qdbqflmf;
    /* fill in the data... */
    strcpy400(fileSpecPtr->qdbqfile, filename,
      sizeof(fileSpecPtr->qdbqfile));
    if (library == NULL)
        strcpy400(fileSpecPtr->qdbqlib, QDBQLIBL,
          sizeof(fileSpecPtr->qdbqlib));
    else
        strcpy400(fileSpecPtr->qdbqlib, library,
          sizeof(fileSpecPtr->qdbqlib));
    if (member == NULL)
        strcpy400(fileSpecPtr->qdbqmbr, QDBQFRST,
          sizeof(fileSpecPtr->qdbqmbr));
    else
        strcpy400(fileSpecPtr->qdbqmbr, member,
          sizeof(fileSpecPtr->qdbqmbr));
    if (format == NULL)
        strcpy400(fileSpecPtr->qdbqfmt, QDBQONLY,
          sizeof(fileSpecPtr->qdbqfmt));
    else
```

```
        strcpy400(fileSpecPtr->qdbqfmt, format,
          sizeof(fileSpecPtr->qdbqfmt));
    /* return the amount of storage used in the file specs */
    return(fileHdr->qdbqfilnum*sizeof(QDBQN_T));
}


/* getRecordFmt(format, format storage size(max),
   file name, file library, file format)
 - get a record format (using QDBRTVFD) */
int getRecordFmt(Qdb_Qddfmt_t *formatPtr, long spaceSize,
  char *filename, char *libname, char *formatname)
{
    error_code errcod;
    char override = '1';  /* process overrides */
    char fileLibname[20];
    char outFilLib[20];
    char format[10];

    if (formatPtr == NULL || filename == NULL)
        return(0);  /* missing data */
    errcod.bytes_provided = 512;
    errcod.msgid[0] = (char) 0;
    /* set up temporary variables... */
    strcpy400(fileLibname, filename, 10);
    if (libname == NULL)
        strcpy400(&fileLibname[10], QDBQLIBL, 10);
    else
        strcpy400(&fileLibname[10], libname, 10);
    if (formatname == NULL)
        strcpy400(format, filename, 10);
    else
        strcpy400(format, formatname, 10);
    /* call the RTVFD API to get the record format */
    QDBRTVFD((char *) formatPtr, spaceSize, outFilLib,
      "FILD0200",
      fileLibname, format, &override,
        "*LCL      ", "*EXT      ", &errcod);
    if (errcod.msgid[0])
    {
        printf("API QDBRTVFD failed\n");
        printf("msgid = %7s\n", strcnv400(errcod.msgid,
          sizeof(errcod.msgid)));
    }
    if (formatPtr->Qddbyrtn != formatPtr->Qddbyava)
        return(0);  /* missing data */
    /* return total storage used in format */
    return(formatPtr->Qddbyrtn);
}


/* copyField(format, field name, file number, existing format)
 - copy a field from an existing format */
long copyField(Qdb_Qddfmt_t *formatPtr, char *fieldName, int fieldFile,
  Qdb_Qddfmt_t *oldFormatPtr)
{
    int i;
    long fieldSize;
    char padField[30];
    Qdb_Qddffld_t *fieldPtr, *oldFieldPtr;

    if (formatPtr == NULL || fieldName == NULL || oldFormatPtr == NULL)
        return(0);  /* missing data */
    strcpy400(padField, fieldName, 30);
    /* set up field pointers */
    fieldPtr = (Qdb_Qddffld_t *) ((char *) formatPtr +
      formatPtr->Qddbyava);
```

```
    oldFieldPtr = (Qdb_Qddffld_t *) (oldFormatPtr + 1);
    /* loop through all the fields, looking for a match */
    for (i=0; i < oldFormatPtr->Qddffldnum; i++,
      oldFieldPtr = (Qdb_Qddffld_t *) ((char *) oldFieldPtr +
      oldFieldPtr->Qddfdefl))
    /* if a match was found... */
    if (memcmp(oldFieldPtr->Qddfflde, padField, 30) == 0)
    {
        /* copy the field over */
        fieldSize = oldFieldPtr->Qddfdefl;
        memcpy(fieldPtr, oldFieldPtr, fieldSize);
        /* set the file number it was defined in */
        fieldPtr->Qddfjref = fieldFile;
        /* increment the format header information */
        formatPtr->Qddffldnum++;
        formatPtr->Qddfrlen += fieldPtr->Qddffldb;
        formatPtr->Qddbyava += fieldSize;
        formatPtr->Qddbyrtn = formatPtr->Qddbyava;
        break;
    }
    /* return total storage used in format */
    return(formatPtr->Qddbyrtn);
}


/* setFieldUsage(format, field name, usage)
 - set the field usage in a format */
void setFieldUsage(Qdb_Qddfmt_t *formatPtr, char *fieldName, char usage)
{
    int i;
    char padField[30];
    Qdb_Qddffld_t *fieldPtr;

    if (formatPtr == NULL)
        return;  /* missing data */
    if (fieldName != NULL)
        strcpy400(padField, fieldName, 30);
    /* set up field pointers */
    fieldPtr = (Qdb_Qddffld_t *) (formatPtr + 1);
    /* loop through all the fields, looking for a match */
    for (i=0; i < formatPtr->Qddffldnum; i++,
      fieldPtr = (Qdb_Qddffld_t *) ((char *) fieldPtr +
      fieldPtr->Qddfdefl))
    /* if all fields to be set or a match was found... */
    if (fieldName == NULL ||
      memcmp(fieldPtr->Qddfflde, padField, 30) == 0)
        fieldPtr->Qddffiob = usage;
}


/* addSelectField(section section, field name, file number for field)
 - add a selection for a file field to the selection section */
int addSelectField(QDBQS_T *selectHdr, char *fieldName, int fieldFile)
{
    QDBQSIT_T *selectItemPtr;
    QDBQSOPF_T *selectFldPtr;
    int itemSize;

    if (selectHdr == NULL || fieldName == NULL)
        return(0);  /* invalid data */
    /* set up all the section for adding a field */
    selectItemPtr = (QDBQSIT_T *) ((char *) selectHdr +
      selectHdr->qdbqsl);
    itemSize = sizeof(QDBQSIT_T) - sizeof(selectItemPtr->qdbqsitm);
    memset((void *) selectItemPtr, (char) 0, itemSize);
    selectFldPtr = (QDBQSOPF_T *) ((char *) selectItemPtr + itemSize);
    memset((void *) selectFldPtr, (char) 0, sizeof(QDBQSOPF_T));
```

```
    /* set up the selection item information for a field */
    selectItemPtr->qdbqslen = itemSize + sizeof(QDBQSOPF_T);
    /* length */
    selectItemPtr->qdbqsitt = QDBQOPF; /* type is field */
    /* now set up the field */
    strcpy400(selectFldPtr->qdbqsofn, fieldName,
      sizeof(selectFldPtr->qdbqsofn));
    selectFldPtr->qdbqsofj = fieldFile;
    /* update the header statistics */
    selectHdr->qdbqsnum++;  /* increment number of select specs */
    selectHdr->qdbqsl += selectItemPtr->qdbqslen;  /* total length */
    /* return the total storage now in the selection section */
    return(selectHdr->qdbqsl);
}


/* addSelectLiteral(selection section, literal, size of literal data)
 - add a selection for a literal to the selection section */
int addSelectLiteral(QDBQS_T *selectHdr, void *literal, int sizeLit)
{
    QDBQSIT_T *selectItemPtr;
    QDBQSOCH_T *selectLitPtr;
    void *selectDataPtr;
    int itemSize;

    if (selectHdr == NULL || literal == NULL || sizeLit < 1)
        return(0);  /* invalid data */
    /* set up all the sections for adding a literal */
    selectItemPtr = (QDBQSIT_T *)
      ((char *) selectHdr + selectHdr->qdbqsl);
    itemSize = sizeof(QDBQSIT_T) - sizeof(selectItemPtr->qdbqsitm);
    memset((void *) selectItemPtr, (char) 0, itemSize);
    selectLitPtr = (QDBQSOCH_T *) ((char *) selectItemPtr + itemSize);
    memset((void *) selectLitPtr, (char) 0, sizeof(QDBQSOCH_T));
    selectDataPtr = (void *) (selectLitPtr + 1);
    /* set up the selection item information for a literal */
    selectItemPtr->qdbqslen = itemSize + sizeof(QDBQSOCH_T) + sizeLit;
    selectItemPtr->qdbqsitt = QDBQOPC;  /* literal type */
    /* now set up the literal */
    selectLitPtr->qdbqsocl = sizeLit;  /* literal size */
    selectLitPtr->qdbqsoft = '\xFF';
      /* use job format for date/time fields */
    memcpy(selectDataPtr, literal, sizeLit);
      /* save the literal value */
    /* update the header statistics */
    selectHdr->qdbqsnum++;  /* increment number of select specs */
    selectHdr->qdbqsl += selectItemPtr->qdbqslen;  /* total length */
    /* return the total storage now in the selection section */
    return(selectHdr->qdbqsl);
}


/* addSelectOperator(selection section, operator type)
 - add a selection for an operator to the selection section */
int addSelectOperator(QDBQS_T *selectHdr, char *operator)
{
    QDBQSIT_T *selectItemPtr;
    QDBQSOPR_T *selectOprPtr;
    QDBQSOP2_T *selectWldPtr;
    int itemSize;
    int oprSize;

    if (selectHdr == NULL || operator == NULL)
        return(0);  /* invalid data */
    /* set up all the sections for adding an operator */
    selectItemPtr = (QDBQSIT_T *)
      ((char *) selectHdr + selectHdr->qdbqsl);
```

```
    itemSize = sizeof(QDBQSIT_T) - sizeof(selectItemPtr->qdbqsitm);
    memset((void *) selectItemPtr, (char) 0, itemSize);
    selectOprPtr = (QDBQSOPR_T *) ((char *) selectItemPtr + itemSize);
    oprSize = sizeof(QDBQSOPR_T) + sizeof(QDBQSOP2_T);
    memset((void *) selectOprPtr, (char) 0, oprSize);
    /* set up the selection item information for an operator */
    selectItemPtr->qdbqslen = itemSize + oprSize;  /* length */
    selectItemPtr->qdbqsitt = QDBQOPTR;  /* operator type */
    /* now set up the operator */
    memcpy(selectOprPtr->qdbqsop, operator,
      sizeof(selectOprPtr->qdbqsop));
    /* wildcard operator set up */
    if (memcmp(operator, QDBQWILD, 2) == 0)
    {
       selectOprPtr->qdbqswc1 = '_';
       selectOprPtr->qdbqswc2 = '*';
       selectWldPtr = (QDBQSOP2_T *) (selectOprPtr + 1);
       memcpy(selectWldPtr->qdbqsdb1,"\42_", 2);
       memcpy(selectWldPtr->qdbqsdb2,"\42*", 2);
    }
    /* update the header statistics */
    selectHdr->qdbqsnum++;  /* increment number of select specs */
    selectHdr->qdbqsl += selectItemPtr->qdbqslen;  /* total length */
    /* return the total storage now in the selection section */
    return(selectHdr->qdbqsl);
}


/* addOrderBy(orderby section, orderby specs section, key field name,
   descend sort option
 - add an order by to the order by section */
int addOrderBy(QDBQKH_T *orderByHdr, QDBQKF_T *orderByFld,
  char *keyfield, int descend)
{
    int i;
    QDBQKF_T *orderByFldPtr;

    if (orderByHdr == NULL || orderByFld == NULL || keyfield == NULL)
        return(0);
    if (orderByHdr->qdbqknum == MAX_ORDERBY)
        return(0);
    /* increment the order by spec counter */
    i = orderByHdr->qdbqknum++;
    /* add the new orderby data */
    orderByFldPtr = &orderByFld[i];
    memset((void *) orderByFldPtr, (char) 0, sizeof(QDBQKF_T));
    strcpy400(orderByFldPtr->qdbqkfld, keyfield,
        sizeof(orderByFldPtr->qdbqkfld));
    orderByFldPtr->qdbqksq.qdbqksad = (descend) ? ON : OFF;
    /* return the space used by the order by specs */
    return(orderByHdr->qdbqknum*sizeof(QDBQKF_T));
}


/* addGroupBy(groupby section, groupby field spec section,
   groupby field name, file number of groupby field)
 - add a group by to the group by section */
int addGroupBy(QDBQGH_T *groupByHdr, QDBQGF_T *groupByFld,
  char *groupfield, int fromFile)
{
    int i;
    QDBQGF_T *groupByFldPtr;

    if (groupByHdr == NULL || groupByFld == NULL || groupfield == NULL)
        return(0);
    if (groupByHdr->qdbqgfnum == MAX_GROUPBY)
        return(0);
```

```
    /* increment the group by spec counter */
    i = groupByHdr->qdbqgfnum++;
    /* add the new groupby data */
    groupByFldPtr = (QDBQGF_T *) &groupByFld[i];
    memset((void *) groupByFldPtr, (char) 0, sizeof(QDBQGF_T));
    strcpy400(groupByFldPtr->qdbqgfld, groupfield,
        sizeof(groupByFldPtr->qdbqgfld));
    groupByFldPtr->qdbqgflj = fromFile;
    /* return the space used by the group by specs */
    return(groupByHdr->qdbqgfnum*sizeof(QDBQGF_T));
}


/* addJoinTest(join section, join test section, join from field name,
   join from file number, join to field name, join to file number,
   join operator)
 - add a join test to the join section */
int addJoinTest(QDBQJHDR_T *joinHdr,
  QDBQJFLD_T *joinSpec, char *fromFld,
  int fromFile, char *toFld, int toFile, char *joinOp)
{
    int i;
    QDBQJFLD_T *joinSpecPtr;

    if (joinHdr == NULL || joinSpec == NULL)
        return(0);
    if (joinHdr->qdbqjknum == MAX_JOINTESTS)
        return(0);
    /* increment the join test counter */
    i = joinHdr->qdbqjknum++;
    memset((void *) &joinSpec[i], (char) 0, sizeof(QDBQJFLD_T));
    /* add the new join data */
    joinSpecPtr = &joinSpec[i];
    strcpy400(joinSpecPtr->qdbqjfnm, fromFld,
      sizeof(joinSpecPtr->qdbqjfnm));
    joinSpecPtr->qdbqjfnum = fromFile;  /* 1, 2, 3, etc */
    strcpy400(joinSpecPtr->qdbqjtnm, toFld,
      sizeof(joinSpecPtr->qdbqjtnm));
    joinSpecPtr->qdbqjtnum = toFile;  /* 1, 2, 3, etc */
    /* Join operator - see #defines in QQ API include */
    strcpy400(joinSpecPtr->qdbqjop, joinOp,
      sizeof(joinSpecPtr->qdbqjop));
    /* set size of entire join spec */
    joinHdr->qdbqjln += sizeof(QDBQJFLD_T);
    /* return the space used by the join tests */
    return(joinHdr->qdbqjknum*sizeof(QDBQJFLD_T));
}


/* addQDTsection(qdt, new section, size of new section, qdt offset)
 - place a new section into the QDT */
void addQDTsection(QDBQH_T *qdtHdr, char *newSection,
  int newSize, int *offset)
{
    char *sectionPtr;

    /* position to the current end of the QDT */
    sectionPtr = (char *) qdtHdr + qdtHdr->qdbspcsize;
    /* append in the new section data */
    memcpy(sectionPtr, newSection, newSize);
    /* if an offset is to be stored, remember it now */
    if (offset != NULL)
        *offset = qdtHdr->qdbspcsize;
    /* update the QDT size */
    qdtHdr->qdbspcsize += newSize;
}
```

```
/* createAccessPlanSpace(access plan, user space name, size)
 - creates a *USRSPC object for storing the access plan */
long createAccessPlanSpace(ACCPLN_T *accessPlan, char *name,
  long spaceSize)
{
    QUSCUSAT_T chgAttr;
    _SPCPTR usrSpcPtr;
    char library[10];
    char value = (char) 0;
    char text[50];
    error_code errcode;

    errcode.bytes_provided = 512;

    strcpy400(text,"Access Plan for QQ API example",50);
    /* Create the User Space */
    QUSCRTUS(name,
             "ACCESSPLAN",
             spaceSize,
             &value,
             "*ALL      ",
             text,
             "*YES      ",
             &errcode,
             "*USER     ");
    if (errcode.msgid[0])
    {
        printf("Create User Space API failed!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(-1);
    }

    /* Change the User Space to allow Auto-Extend */
    strcpy400(library,&name[10],10);
    chgAttr.numAttrs = 1;
    chgAttr.key = 3; /* Auto extend */
    chgAttr.length = sizeof(char);
    chgAttr.data.autoExtend = '1';
    QUSCUSAT(library,
             name,
             &chgAttr,
             &errcode);
    if (errcode.msgid[0])
    {
        printf("Change User Space Attributes FAILED!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(-1);
    }

    /* Retrieve Space Pointer to the User Space */
    QUSPTRUS(name,
             &usrSpcPtr,
             &errcode);
    if (errcode.msgid[0])
    {
        printf("Retrieve Space Pointer to User Space FAILED!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(-1);
    }
    /* Now move to the access plan itself (on 16 byte boundary) */
    accessPlan->storagePtr = (_SPCPTR) ((char*) usrSpcPtr + 16);

    return(0);
```

```
}


/* saveAccessPlan(access plan)
 - update the size in the access plan (QQQQRY actually wrote the data) */
int saveAccessPlan(ACCPLN_T *accessPlan)
{
    _SPCPTR usrSpcPtr;

    /* Position to the start of the user space */
    usrSpcPtr = (_SPCPTR) ((char*) accessPlan->storagePtr - 16);
    /* Write the access plan size out at the start */
    memcpy(usrSpcPtr, (void *) &accessPlan->size,
      sizeof(accessPlan->size));
#ifdef QQDEBUG
    printf("AP size = %ld\n", accessPlan->size);
#endif
    return(0);
}


/* saveQDT(qdt, access plan)
 - append the QDT to the end of the access plan */
int saveQDT(QDBQH_T *qdtPtr, ACCPLN_T *accessPlan)
{
    _SPCPTR usrSpcPtr;

    /* Position to the just after the access plan */
    usrSpcPtr = (_SPCPTR) ((char*) accessPlan->storagePtr +
      accessPlan->size);
    /* Write the QDT size out */
    memcpy(usrSpcPtr, &qdtPtr->qdbspcsize, sizeof(qdtPtr->qdbspcsize));
#ifdef QQDEBUG
    printf("qdt size = %ld\n", qdtPtr->qdbspcsize);
#endif
    /* Move up the user space pointer */
    usrSpcPtr = (_SPCPTR) ((char *) usrSpcPtr + 16);
    /* Write the QDT itself out */
    memcpy(usrSpcPtr, qdtPtr, qdtPtr->qdbspcsize);
    return(0);
}


/* loadQDT(qdt, access plan)
 - load the QDT from the end of the access plan */
int loadQDT(QDBQH_T *qdtPtr, ACCPLN_T *accessPlan)
{
    _SPCPTR usrSpcPtr;

    /* Position to the just after the access plan */
    usrSpcPtr = (_SPCPTR) ((char*) accessPlan->storagePtr +
      accessPlan->size);
    /* Write the QDT size out */
    memcpy((void *) &qdtPtr->qdbspcsize, usrSpcPtr,
      sizeof(qdtPtr->qdbspcsize));
#ifdef QQDEBUG
    printf("qdt size = %ld\n", qdtPtr->qdbspcsize);
#endif
    /* Move up the user space pointer */
    usrSpcPtr = (_SPCPTR) ((char *) usrSpcPtr + 16);
    /* Write the QDT itself out */
    memcpy((void *) qdtPtr, usrSpcPtr, qdtPtr->qdbspcsize);
    return(qdtPtr->qdbspcsize);
}


/* loadAccessPlan(access plan, userspace name)
```

```
 - loads an access plan from a *USRSPC object */
long loadAccessPlan(ACCPLN_T *accessPlan, char *name)
{
    Qus_SPCA_0100_t usrSpcAttr;
    _SPCPTR usrSpcPtr;
    error_code errcode;

    errcode.bytes_provided = 512;
    errcode.msgid[0] = (char) 0;

    /* Retrieve Space Pointer to the User Space */
    QUSPTRUS(name, &usrSpcPtr, &errcode);
    if (errcode.msgid[0])
    {
        printf("Retrieve Space Pointer to User Space FAILED!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(0);
    }

    /* Retrieve Size of Access Plan */
    QUSRUSAT(&usrSpcAttr,
            sizeof(Qus_SPCA_0100_t),
            "SPCA0100",
            name,
            &errcode);
    if (errcode.msgid[0])
    {
        printf("Retrieve User Space Attributes FAILED!\n");
        printf("msgid = %7s\n", strcnv400(errcode.msgid,
          sizeof(errcode.msgid)));
        return(0);
    }
#ifdef QQDEBUG
    else
    {
        printf("Original User Space Attributes\n");
        printf("Bytes Returned  ==> %d\n",usrSpcAttr.Bytes_Returned);
        printf("Bytes Available ==> %d\n",usrSpcAttr.Bytes_Available);
        printf("Space Size      ==> %d\n",usrSpcAttr.Space_Size);
        printf("Auto Extend     ==> %c\n",
          usrSpcAttr.Automatic_Extendability);
    }
#endif
    /* Pull the access plan size out first */
    memcpy((void *) &accessPlan->size, usrSpcPtr,
      sizeof(accessPlan->size));
#ifdef QQDEBUG
    printf("AP size = %ld\n", accessPlan->size);
#endif
    /* Now move to the access plan itself (on 16 byte boundary) */
    accessPlan->storagePtr = (_SPCPTR) ((char*) usrSpcPtr + 16);

    return(accessPlan->size);
}


/*********************************************************************/
```

Defining a simple query

This simple query is equivalent to an SQL query:

```
SELECT * FROM OPENFILE1
ORDER BY LNAME
```

```
/*****************************************************************/
/*  PROGRAM:  QQAPI1                                             */
/*                                                               */
/*  LANGUAGE:  ILE C                                 */
/*                                                               */
/*  DESCRIPTION:  THIS PROGRAM DEFINES A SIMPLE QUERY TO PERFORM    */
/*    ORDERING.                                                  */
/*                                                               */
/*  APIs USED:  QQQQRY                                           */
/*                                                               */
/*****************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <recio.h>
#include <qdbrtvfd.h>
#include <qqqqry.h>
#include "qqapi.h"

/* get the record format from the file */
#pragma mapinc("recfmt","APIQQ/OPENFILE1(OPENFILE1)","input","p z",,)
#include "recfmt"



/* main - start of the program
 *
 * Flow:
 * - initialize variables
 * - override to set up sharing
 * - build various QDT sections
 * - build QDT with those sections
 * - QQQQRY to run the query
 * - open the data path
 * - read the data and display it
 * - close the data paths
 *
 */
main()
{
    /* record I/O variables */
    _RIOFB_T *feedback;
    _RFILE *file1;
    APIQQ_OPENFILE1_OPENFILE1_i_t recBuf;
    int recCount = 0;

    /* Query variables */
    QDBUFCB_T ufcbBuf;
    char qdtBuf[QDT_SIZE];
    char formatBuf[FORMAT_SIZE];
    QDBQH_T *qdtPtr;
    Qdb_Qddfmt_t *formatPtr;
    QDBQFHDR_T fileHdr;
    QDBQN_T fileSpec[MAX_FILES];
    QDBQKH_T orderByHdr;
    QDBQKF_T orderByFld[MAX_ORDERBY];
    int formatSize;
    int fileSpecSize;
    int orderBySize;
    error_code errcod;


    errcod.bytes_provided = 512;
    /* initialize the pointers */
    qdtPtr = (QDBQH_T *) qdtBuf;
```

```
formatPtr = (Qdb_Qddfmt_t *) formatBuf;

/* initialize the headers */
initQDT(qdtPtr, QDBQTEMO, ON, ON, QDBQFINA, 0);
initFile(&fileHdr, QDBQINNJ, QDBQMFOA);
initOrderBy(&orderByHdr);

/* set up override to allow sharing */
system("OVRDBF FILE(OPENFILE1) SHARE(*YES)");
/* Note:  If level checking is not done
   (ie. no format on initUFCB) then
   the override above must specify LVLCHK(*NO) */

/* build the individual QDT sections */
fileSpecSize = addFile(&fileHdr, fileSpec, "OPENFILE1",
  NULL, NULL, NULL);
formatSize = getRecordFmt(formatPtr, FORMAT_SIZE, "OPENFILE1",
  NULL, NULL);
orderBySize = addOrderBy(&orderByHdr, orderByFld, "LNAME", OFF);

/* initialize the UFCB */
initUFCB(&ufcbBuf, QO_INPUT, formatPtr);

/* Now build the real QDT... */
addQDTsection(qdtPtr, (char *) &fileHdr,
  sizeof(fileHdr), &qdtPtr->qdbqfilo);
addQDTsection(qdtPtr, (char *) fileSpec, fileSpecSize, NULL);
addQDTsection(qdtPtr, (char *) formatPtr,
  formatSize, &qdtPtr->qdbqfldo);
addQDTsection(qdtPtr, (char *) &orderByHdr, sizeof(orderByHdr),
  &qdtPtr->qdbqkeyo);
addQDTsection(qdtPtr, (char *) orderByFld, orderBySize, NULL);

/* Finally, run the query! */
QQQQRY("RUNQRY    ", (char *) &ufcbBuf, qdtBuf, NULL, NULL,
  &errcod);
if (errcod.msgid[0])
{
    printf("API QQQQRY failed\n");
    printf("msgid = %7s\n", strcnv400(errcod.msgid,
      sizeof(errcod.msgid)));
}
/* Now access the data */
if ((file1 = _Ropen("OPENFILE1", "rr riofb=N")) == NULL)
{
    printf("Error opening file\n");
    exit(1);
}

/* Perform any record I/O here... */
_Rformat(file1, "OPENFILE1 ");
printf("First name   Last name         State\n");
feedback = _Rreadn(file1, (void *) &recBuf, sizeof(recBuf), __DFT);
while (feedback->num_bytes == sizeof(recBuf))
{
   recCount++;
   printf("%s   ", strcnv400(recBuf.FNAME, sizeof(recBuf.FNAME)));
   printf("%s   ", strcnv400(recBuf.LNAME, sizeof(recBuf.LNAME)));
   printf("%s\n",  strcnv400(recBuf.STATE, sizeof(recBuf.STATE)));
   feedback = _Rreadn(file1, (void *) &recBuf,
     sizeof(recBuf), __DFT);
}
printf("%d records selected\n", recCount);

/* Close the file */
_Rclose(file1);
```

```
    /* close out the QDT file handle */
    system("RCLRSC");
}
```

Defining a join query

This join query is equivalent to an SQL query:

```
SELECT * FROM OPENFILE1 A, OPENFILE2 B
WHERE STATE = 'AK' AND
      A.ACCTNUM = B.CUSTNUM
```

```
/*********************************************************************/
/*  PROGRAM:  QQAPI7                                                 */
/*                                                                   */
/*  LANGUAGE:  ILE C                                                 */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM DEFINES A JOIN QUERY.                 */
/*                                                                   */
/*  APIs USED:  QQQQRY                                               */
/*                                                                   */
/*********************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <recio.h>
#include <qdbrtvfd.h>
#include <qqqqry.h>
#include "qqapi.h"

/* get the record format from the file */
#pragma mapinc("recfmt","APIQQ/FORMAT1(FORMAT1)","input","p z",,)
#include "recfmt"



/* main - start of the program
 *
 * Flow:
 * - initialize variables
 * - override to set up sharing
 * - build various QDT sections
 * - build QDT with those sections
 * - QQQQRY to run the query
 * - open the data path
 * - read the data and display it
 * - close the data paths
 *
 */
main()
{
    /* record I/O variables */
    _RIOFB_T *feedback;
    _RFILE *file1;
    APIQQ_FORMAT1_FORMAT1_i_t recBuf;
    int recCount = 0;

    /* Query variables */
    QDBUFCB_T ufcbBuf;
    char qdtBuf[QDT_SIZE];
    char formatBuf[FORMAT_SIZE];
    char selectBuf[SELECT_SIZE];
    QDBQH_T *qdtPtr;
    Qdb_Qddfmt_t *formatPtr;
    QDBQS_T *selectPtr;
```

```
QDBQFHDR_T fileHdr;
QDBQN_T fileSpec[MAX_FILES];
QDBQJHDR_T joinHdr;
QDBQJFLD_T joinSpec[MAX_JOINTESTS];
int formatSize;
int fileSpecSize;
int selectSize;
int joinSize;
error_code errcod;


errcod.bytes_provided = 512;
/* initialize the pointers */
qdtPtr = (QDBQH_T *) qdtBuf;
formatPtr = (Qdb_Qddfmt_t *) formatBuf;
selectPtr = (QDBQS_T *) selectBuf;

/* initialize the headers */
initQDT(qdtPtr, QDBQTEMO, ON, ON, QDBQFINA, 0);
initFile(&fileHdr, QDBQINNJ, QDBQMFOA);
initSelection(selectPtr);
initJoin(&joinHdr);

/* set up override to allow sharing */
system("OVRDBF FILE(OPENFILE1) SHARE(*YES) LVLCHK(*NO)");
/* Note:  If level checking is not done
   (ie. no format on initUFCB) then
   the override above must specify LVLCHK(*NO) */

/* build the individual QDT sections */
addFile(&fileHdr, fileSpec, "OPENFILE1", NULL, NULL, NULL);
fileSpecSize = addFile(&fileHdr, fileSpec, "OPENFILE2",
  NULL, NULL, NULL);
formatSize = getRecordFmt(formatPtr, FORMAT_SIZE, "FORMAT1",
  NULL, NULL);
joinSize = addJoinTest(&joinHdr, joinSpec, "ACCTNUM", 1,
  "CUSTNUM" , 2, "EQ");
/* build selection test:  STATE = 'AK' */
addSelectField(selectPtr, "STATE", 1);
addSelectLiteral(selectPtr, "'AK'", 4);
selectSize = addSelectOperator(selectPtr, QDBQEQ);

/* initialize the UFCB */
initUFCB(&ufcbBuf, QO_INPUT, NULL);

/* Now build the real QDT... */
addQDTsection(qdtPtr, (char *) &fileHdr,
  sizeof(fileHdr), &qdtPtr->qdbqfilo);
addQDTsection(qdtPtr, (char *) fileSpec, fileSpecSize, NULL);
addQDTsection(qdtPtr, (char *) formatPtr,
  formatSize, &qdtPtr->qdbqfldo);
addQDTsection(qdtPtr, (char *) &joinHdr,
  sizeof(joinHdr),&qdtPtr->qdbqjoio);
addQDTsection(qdtPtr, (char *) joinSpec, joinSize, NULL);
addQDTsection(qdtPtr, (char *) selectPtr,
  selectSize, &qdtPtr->qdbqselo);

/* Finally, run the query! */
QQQQRY("RUNQRY    ", (char *) &ufcbBuf, qdtBuf, NULL, NULL,
  &errcod);
if (errcod.msgid[0])
{
    printf("API QQQQRY failed\n");
    printf("msgid = %7s\n", strcnv400(errcod.msgid,
      sizeof(errcod.msgid)));
}
/* Now access the data */
```

```
    if ((file1 = _Ropen("OPENFILE1", "rr riofb=N")) == NULL)
    {
        printf("Error opening file\n");
        exit(1);
    }

    /* Perform any record I/O here... */
    _Rformat(file1, "FORMAT1");
    printf("Last name        Item name\n");
    feedback = _Rreadn(file1, (void *) &recBuf, sizeof(recBuf), __DFT);
    while (feedback->num_bytes == sizeof(recBuf))
    {
       recCount++;
       printf("%s  ", strcnv400(recBuf.LNAME, sizeof(recBuf.LNAME)));
       printf("%s\n", strcnv400(recBuf.ITEMNAME,
         sizeof(recBuf.ITEMNAME)));
       feedback = _Rreadn(file1, (void *) &recBuf,
         sizeof(recBuf), __DFT);
    }
    printf("%d records selected\n", recCount);

    /* Close the file */
    _Rclose(file1);

    /* close out the QDT file handle */
    system("RCLRSC");
}
```

Defining a join query with selection, grouping, and ordering

This join query with selection, grouping, and ordering is equivalent to an SQL query:

```
SELECT LNAME, FNAME, ITEMCODE, ITEMNAME, STATUS
FROM OPENFILE1, OPENFILE2
WHERE STATE = 'AK' AND CUSTNUM = ACCTNUM
GROUP BY LNAME, FNAME, ITEMCODE, ITEMNAME, STATUS
ORDER BY ITEMNAME
```

```
/*********************************************************************/
/*  PROGRAM:  QQAPI11                                                */
/*                                                                   */
/*  LANGUAGE:  ILE C                                                 */
/*                                                                   */
/*  DESCRIPTION:  THIS PROGRAM DEFINES A JOIN QUERY WITH SELECTION   */
/*    GROUPING AND ORDERING.                                         */
/*                                                                   */
/*  APIs USED:  QQQQRY                                               */
/*                                                                   */
/*********************************************************************/


#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <qdbrtvfd.h>
#include <qqqqry.h>
#include "qqapi.h">



/* main - start of the program
 *
 * Flow:
 * - initialize variables
 * - override to set up sharing
 * - build various QDT sections
 * - build QDT with those sections
 * - QQQQRY to run the query
```

```
 * - open the data path
 * - read the data and display it
 * - close the data paths
 *
 */
main()
{
    /* file I/O variables */
#define REC_SIZE  52
    FILE *file1;
    char recBuf[REC_SIZE];
    int recCount = 0, found;

    /* Query variables */
    QDBUFCB_T ufcbBuf;
    char qdtBuf[QDT_SIZE];
    char formatBuf[FORMAT_SIZE];
    char tempFormatBuf[FORMAT_SIZE];
    char selectBuf[SELECT_SIZE];
    QDBQH_T *qdtPtr;
    Qdb_Qddfmt_t *formatPtr;
    Qdb_Qddfmt_t *tempFormatPtr;
    QDBQS_T *selectPtr;
    QDBQFHDR_T fileHdr;
    QDBQN_T fileSpec[MAX_FILES];
    QDBQJHDR_T joinHdr;
    QDBQJFLD_T joinSpec[MAX_JOINTESTS];
    QDBQKH_T orderByHdr;
    QDBQGH_T groupByHdr;
    QDBQKF_T orderByFld[MAX_ORDERBY];
    QDBQGF_T groupByFld[MAX_GROUPBY];
    int formatSize;
    int fileSpecSize;
    int orderBySize;
    int groupBySize;
    int selectSize;
    int joinSize;
    error_code errcod;

    memset( (void *) &errcod, (char) 0, sizeof(error_code) );
    errcod.bytes_provided = 512;
    /* initialize the pointers */
    qdtPtr = (QDBQH_T *) qdtBuf;
    formatPtr = (Qdb_Qddfmt_t *) formatBuf;
    tempFormatPtr = (Qdb_Qddfmt_t *) tempFormatBuf;
    selectPtr = (QDBQS_T *) selectBuf;

    /* initialize the headers */
    initQDT(qdtPtr, QDBQTEMO, ON, ON, QDBQFINA, 0);
    initFile(&fileHdr, QDBQINNJ, QDBQMFOA);
    initFormat(formatPtr, "JOINFMT01");
    initOrderBy(&orderByHdr);
    initGroupBy(&groupByHdr);
    initSelection(selectPtr);
    initJoin(&joinHdr);

    /* set up override to allow sharing */
    system("OVRDBF FILE(OPENFILE1) SHARE(*YES) LVLCHK(*NO)");
    /* Note:  If level checking is not done
       (ie. no format on initUFCB) then
       the override above must specify LVLCHK(*NO) */

    /* build the individual QDT sections */
    addFile(&fileHdr, fileSpec, "OPENFILE1", NULL, NULL, NULL);
    fileSpecSize = addFile(&fileHdr, fileSpec, "OPENFILE2",
      NULL, NULL, NULL);
    /* get the first format and copy some fields */
```

```
getRecordFmt(tempFormatPtr, FORMAT_SIZE, "OPENFILE1",
  NULL, NULL);
copyField(formatPtr, "LNAME", 1, tempFormatPtr);
copyField(formatPtr, "FNAME", 1, tempFormatPtr);
/* clear the old format data */
memset(tempFormatPtr, 0, FORMAT_SIZE);
/* get the second format and copy some more fields */
getRecordFmt(tempFormatPtr, FORMAT_SIZE, "OPENFILE2", NULL, NULL);
copyField(formatPtr, "ITEMCODE", 2, tempFormatPtr);
copyField(formatPtr, "ITEMNAME", 2, tempFormatPtr);
formatSize = copyField(formatPtr, "STATUS", 2, tempFormatPtr);
/* set all the fields to input only */
setFieldUsage(formatPtr, NULL, 1);
/* build selection test:  STATE = 'AK' */
addSelectField(selectPtr, "STATE", 1);
addSelectLiteral(selectPtr, "'AK'", 4);
selectSize = addSelectOperator(selectPtr, QDBQEQ);
joinSize = addJoinTest(&joinHdr, joinSpec, "ACCTNUM", 1,
  "CUSTNUM" , 2, "EQ");
orderBySize = addOrderBy(&orderByHdr, orderByFld,
  "ITEMNAME", OFF);
addGroupBy(&groupByHdr, groupByFld, "LNAME", 0);
addGroupBy(&groupByHdr, groupByFld, "FNAME", 0);
addGroupBy(&groupByHdr, groupByFld, "ITEMCODE", 0);
addGroupBy(&groupByHdr, groupByFld, "ITEMNAME", 0);
groupBySize = addGroupBy(&groupByHdr, groupByFld, "STATUS", 0);


/* initialize the UFCB */
initUFCB(&ufcbBuf, QO_INPUT, NULL);
/* set up for sequential only processing since it is a group by */
ufcbBuf.qufcb.parameter.seqonly = SEQUPROC;
ufcbBuf.qufcb.parameter.seqonoff = ON;
ufcbBuf.qufcb.parameter.numonoff = ON;
ufcbBuf.qufcb.parameter.numrecs = 1;

/* Now build the real QDT... */
addQDTsection(qdtPtr, (char *) &fileHdr,
  sizeof(fileHdr), &qdtPtr->qdbqfilo);
addQDTsection(qdtPtr, (char *) fileSpec, fileSpecSize, NULL);
addQDTsection(qdtPtr, (char *) formatPtr,
  formatSize, &qdtPtr->qdbqfldo);
addQDTsection(qdtPtr, (char *) &joinHdr,
  sizeof(joinHdr),&qdtPtr->qdbqjoio);
addQDTsection(qdtPtr, (char *) joinSpec, joinSize, NULL);
addQDTsection(qdtPtr, (char *) selectPtr,
  selectSize, &qdtPtr->qdbqselo);
addQDTsection(qdtPtr, (char *) &orderByHdr, sizeof(orderByHdr),
  &qdtPtr->qdbqkeyo);
addQDTsection(qdtPtr, (char *) orderByFld, orderBySize, NULL);
addQDTsection(qdtPtr, (char *) &groupByHdr, sizeof(groupByHdr),
  &qdtPtr->qdbqgrpo);
addQDTsection(qdtPtr, (char *) groupByFld, groupBySize, NULL);

/* Finally, run the query! */
QQQQRY("RUNQRY    ", (char *) &ufcbBuf, qdtBuf,
  NULL, NULL, &errcod);
if (errcod.msgid[0])
{
    printf("API QQQQRY failed\n");
    printf("msgid = %7s\n", strcnv400(errcod.msgid,
      sizeof(errcod.msgid)));
}
/* Now access the data */
if ((file1 = fopen("OPENFILE1", "rb")) == NULL)
{
    printf("Error opening file\n");
```

```
        exit(1);
    }

    /* Perform any record I/O here... */
    printf("Last name        First name  Code   \
Item                  St\n");
    found = fread((void *) &recBuf, REC_SIZE, 1, file1);
    while (found)
    {
       recCount++;
       printf("%s  ", strcnv400(recBuf, 15));
       printf("%s  ", strcnv400(&recBuf[15], 10));
       printf("%s  ", strcnv400(&recBuf[25], 5));
       printf("%s  ", strcnv400(&recBuf[30], 20));
       printf("%s\n", strcnv400(&recBuf[50], 2));
       found = fread((void *) &recBuf, REC_SIZE, 1, file1);
    }
    printf("%d records selected\n", recCount);

    /* Close the file */
    fclose(file1);

    /* close out the QDT file handle */
    system("RCLRSC");
}
```

Example: Deleting old spooled files

In this example, a user-defined command Delete Old Spooled Files (DLTOLDSPLF) is created, which calls a program named DLTOLDSPLF to delete a list of old spooled files.

This example has three major parts:

1. The DLTOLDSPLF command calls the delete old spooled files (DLTOLDSPLF) program in one of the following languages:
   - OPM RPG
   - OPM COBOL
   - ILE C
2. The DLTOLDSPLF program is supplied in OPM RPG, OPM COBOL, and ILE C. It performs the following operations:
   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 APIs:
      - 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 performs the following operations:
   a. Deletes the specified spooled files (DLTSPLF command).
   b. Sends a message if the spooled file was deleted (SNDPGMMSG command).

Notes:

- The programs and source code used as examples in the spooled file portion of this topic exist only in printed form. They are not stored electronically on the system.

- By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

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:')
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/DLTOLDSPLF) +
    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
- ILE C

RPG DLTOLDSPLF program

To delete old spooled files, use the following RPG program:

```
     H* ****************************************************************
     H* ****************************************************************
     H*                                                                *
     H* MODULE:    DLTOLDSPLF                                          *
     H*                                                                *
     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* ************************************************************
E/COPY QRPGSRC,EUSRSPLA
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
ITGTDAT      DS
I                                       1   1 TGTCEN
I                                       2   3 TGTYR
I                                       4   5 TGTMTH
I                                       6   7 TGTDAY
I/COPY QRPGSRC,QUSGEN
I/COPY QRPGSRC,QUSLSPL
I/COPY QRPGSRC,QUSRSPLA
I****************************************************************
I* The following is copied from QSYSINC/QRPGSRC member QUSEC
I* so that the variable length field QUSBNG can be defined
I* as 100 bytes for exception data.  The defined field is
I* named EXCDTA.
I****************************************************************
IQUSBN       DS
I*                                            Qus EC
I                                   B   1   40QUSBNB
I*                                            Bytes Provided
I                                   B   5   80QUSBNC
I*                                            Bytes Available
I                                       9  15 QUSBND
I*                                            Exception Id
I                                      16  16 QUSBNF
I*                                            Reserved
I*                                     17  17 QUSBNG
I*                                     Varying length
I                                      17 116 EXCDTA
IDATSTR      DS
I                                       1   1 DATCEN
I                                       2   3 DATYR
I                                       4   5 DATMTH
I                                       6   7 DATDAY
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   QUSBN
C                     Z-ADD0         QUSBNB
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           QUSBN
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           QUSBN
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-ADD140       GENLEN
C                     Z-ADD1         STRPOS
C*                                                              *
C                     CALL 'QUSRTVUS'
C                     PARM           USRSPC
C                     PARM           STRPOS
C                     PARM           GENLEN
C                     PARM           QUSBP
C                     PARM           QUSBN
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-ADDQUSBPQ    STRPOS
C                     ADD  1         STRPOS
C                     Z-ADDQUSBPT    RTVLEN
C                     Z-ADD209       RCVLEN
C                     Z-ADD1         COUNT  150
C*                                                              *
C* ************************************************************
C* ************************************************************
C*                                                              *
C*     BEGINNING OF LOOP (DO WHILE COUNT <= QUSBPS)             *
C*                                                              *
C* ************************************************************
C*                                                              *
C           COUNT     DOWLEQUSBPS
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           QUSFT
C                     PARM           QUSBN
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 QUSFTH    QUSFXD
C                     MOVE QUSFTJ    QUSFXF
C                     MOVEL'*INT'    SPLFNM 10
C                     MOVE *BLANKS   SPLF#
C*                                                             *
C                     CALL 'QUSRSPLA'
C                     PARM           QUSFX
C                     PARM           RCVLEN
C                     PARM 'SPLA0100'FMTNM2  8
C                     PARM           JOBINF
C                     PARM           QUSFXD
C                     PARM           QUSFXF
C                     PARM           SPLFNM
C                     PARM           SPLF#
C                     PARM           QUSBN
C*                                                             *
C* CHECK QUSFX 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                     MOVE QUSFX7    DATSTR

C           DATCEN    IFLT TGTCEN
C                     EXSR CLDLT
C                     ELSE
C           DATCEN    IFEQ TGTCEN

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                     END                           FOR EQ CEN
C                     END                           FOR ELSE CEN

C*                                                             *
C* GO  BACK AND PROCESS THE REST OF THE ENTRIES IN THE USER    *
C* SPACE.                                                      *
C           QUSBPT    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           QUSBN
C*                                                              *
C* DELETE THE USER SPACE OBJECT THAT WAS CREATED.               *
C*                                                              *
C                   CALL 'QUSDLTUS'
C                   PARM           USRSPC
C                   PARM           QUSBN
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 QUSFXL    FIL#
C                   CALL 'CLDLT'
C                   PARM           QUSFXK
C                   PARM           QUSFXJ
C                   PARM           QUSFXH
C                   PARM           QUSFXG
C                   PARM           FIL#
C                   PARM           QUSFXM
C                   PARM           QUSFXN
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.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 FILE SECTION.
 WORKING-STORAGE SECTION.
 COPY QUSGEN OF QSYSINC-QLBLSRC.
 COPY QUSLSPL OF QSYSINC-QLBLSRC.
 COPY QUSRSPLA OF QSYSINC-QLBLSRC.
*****************************************************************
* VALUES USED FOR ERROR CODE                                    *
*****************************************************************
* The following is copied from QSYSINC/QLBLSRC member QUSEC
* so that the variable length field EXCEPTION-DATA can be defined
* as 100 bytes for exception data.
*****************************************************************
 01  QUS-EC.
     05  BYTES-PROVIDED                     PIC S9(00009) BINARY.
     05  BYTES-AVAILABLE                    PIC S9(00009) BINARY.
     05  EXCEPTION-ID                       PIC  X(00007).
     05  RESERVED                           PIC  X(00001).
*    05  EXCEPTION-DATA                     PIC  X(00001).
*
*                                    Varying length
     05 EXCEPTION-DATA          PIC X(100).

*****************************************************************
* 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 QUSRTVUS PROGRAM                          *
*****************************************************************
 01  RTV-START-POS              PIC S9(9) BINARY VALUE 1.
 01  RTV-LENGTH                 PIC S9(9) BINARY VALUE 140.

 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-YEAR                  PIC X(2).
     05 R-MONTH                 PIC X(2).
     05 R-DAY                   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  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-YEAR                 PIC X(2).
     05 P-MONTH                PIC X(2).
     05 P-DAY                  PIC X(2).

*****************************************************************
* BEGINNING OF EXECUTABLE CODE.                                 *
*****************************************************************
 PROCEDURE DIVISION USING PARM-USERNAME,
                          PARM-OUTQ,
                          PARM-DATE.

 MAIN-PROGRAM.

*    ************************************************************
*    * INITIALIZE ERROR CODE STRUCTURE.                        *
*    ************************************************************

      MOVE 116 TO BYTES-PROVIDED.
      MOVE 0 TO BYTES-AVAILABLE.
      MOVE SPACES TO EXCEPTION-ID.
      MOVE SPACES TO RESERVED OF QUS-EC.
      MOVE SPACES TO EXCEPTION-DATA.

*    ************************************************************
*    * 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,
                           QUS-EC.

*    ********************************************************
*    * LIST THE SPOOLED FILES TO THE USER SPACE OBJECT.       *
*    ********************************************************

     CALL "QUSLSPL"  USING CRT-SPCNAME,
                           LSPLA-FORMAT,
                           PARM-USERNAME,
                           PARM-OUTQ,
                           LSPLA-FORMTYPE,
                           LSPLA-USERDATA,
                           QUS-EC.

*    ********************************************************
*    * RETRIEVE ENTRY INFORMATION FROM THE USER SPACE.        *
*    ********************************************************

     CALL "QUSRTVUS" USING CRT-SPCNAME,
                           RTV-START-POS,
                           RTV-LENGTH,
                           QUS-GENERIC-HEADER-0100,
                           QUS-EC.

*    ********************************************************
*    * IF ANY SPOOLED FILES WERE FOUND MATCHING THE SEARCH    *
*    * CRITERIA, RETRIEVE DETAILED INFORMATION AND DECIDE     *
*    * WHETHER TO DELETE THE FILE OR NOT.                     *
*    ********************************************************

     IF NUMBER-LIST-ENTRIES OF QUS-GENERIC-HEADER-0100
                            GREATER THAN ZERO THEN
         ADD 1 TO OFFSET-LIST-DATA OF QUS-GENERIC-HEADER-0100
                            GIVING RTV-START-POS.
         PERFORM CHECK-AND-DELETE THROUGH
             CHECK-AND-DELETE-END NUMBER-LIST-ENTRIES
                            OF QUS-GENERIC-HEADER-0100 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,
                           QUS-EC.

     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,
                          QUS-EC.
```

```
     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,
                           SIZE-EACH-ENTRY OF
                                       QUS-GENERIC-HEADER-0100,
                           QUS-SPLF0100,
                           QUS-EC.

*    ********************************************************
*    * ADVANCE TO NEXT SPOOLED FILE FOR PROCESSING THE CHECK  *
*    * AND DELETE.                                           *
*    ********************************************************

     ADD SIZE-EACH-ENTRY OF QUS-GENERIC-HEADER-0100 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 QUS-SPLA0100,
                           SPLA-VAR-LENGTH,
                           RSPLA-FORMAT,
                           RSPLA-JOB-NAME,
                           INT-JOB-ID OF QUS-SPLF0100,
                           INT-SPLF-ID OF QUS-SPLF0100,
                           RSPLA-NAME,
                           RSPLA-NUMBER,
                           QUS-EC.

     MOVE DATE-FILE-OPEN OF QUS-SPLA0100 TO RSPLA-DATE.

*    ********************************************************
*    * COMPARE THE CREATE DATE WITH THE DATE THAT WAS PASSED  *
*    * IN AS PARAMETER.                                      *
*    ********************************************************

     IF R-CENTURY IS LESS THAN P-CENTURY THEN
       PERFORM DLT-SPLF THROUGH DLT-SPLF-END
     ELSE
       IF R-CENTURY IS EQUAL TO P-CENTURY THEN

     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 SPLF-NUMBER OF QUS-SPLA0100 TO DLT-SPL-NUMBER.

           CALL "CLDLT" USING SPLF-NAME OF QUS-SPLA0100,
                              JOB-NUMBER OF QUS-SPLA0100,
                              USR-NAME OF QUS-SPLA0100,
                              JOB-NAME OF QUS-SPLA0100,
                              DLT-SPL-NUMBER,
                              FORM-TYPE OF QUS-SPLA0100,
                              USR-DATA OF QUS-SPLA0100.

       DLT-SPLF-END.
```

To create the COBOL program, specify the following:

```
CRTCBLPGM PGM(QGPL/DLTOLDSPLF) SRCFILE(QGPL/QCBLSRC)
```

ILE C DLTOLDSPLF program

To delete spooled files, you can use this ILE C DLTOLDSPLF program:

```
/*****************************************************************/
/*  PROGRAM:  DLTOLDSPLF                                         */
/*                                                               */
/*  LANGUAGE: ILE C                                              */
/*                                                               */
/*  DESCRIPTION:  THIS IS AN EXAMPLE PROGRAM FOR THE USE OF      */
/*                USER SPACES WRITTEN IN ILE C.                  */
/*                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 <stdio.h>
#include <qusec.h>      /*Error code structures                 */
#include <qusgen.h>     /*General user space structures         */
#include <quscrtus.h>   /*Linkage info, structures for QUSCRTUS  */
#include <quslspl.h>    /*Linkage info, structures for QUSLSPL   */
#include <qusptrus.h>   /*Linkage info, structures for QUSPTRUS  */
#include <qusrspla.h>   /*Linkage info, structures for QUSRSPLA  */
#include <qusdltus.h>   /*Linkage info, structures for QUSDLTUS  */
#include <qmhsndm.h>    /*Linkage info, structures for QMHSNDM   */
#include <qmhsndpm.h>   /*Linkage info, structures for QMHSNDPM  */


#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);

Qus_Generic_Header_0100_t *space;
char *list_section;
Qus_SPLF0100_t *entry_list;
Qus_SPLA0100_t *Rcv_Spl_Var;
/*****************************************************************/
/* PARMS FOR CLDLT                                               */
/*****************************************************************/
char job_nmbr[6];
char usr_nm[10];
char job_nm[10];
char sp_job_name[10];
char sp_spl_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 tmp_spl_number[7];
char dlt_date[7];
char spc_date[7];
int  api_code;
Qus_EC_t err_code;

/*******************************************************************/
/* PROCEDURE TO CHECK THE ERRCODE RETURNED FROM CALLS TO APIs    */
/*******************************************************************/
void error_check(void)
{
if (err_code.Bytes_Available != 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[];
{

/*******************************************************************/
/*  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_Provided = 0;
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->Offset_List_Data;
entry_list = (Qus_SPLF0100_t *) list_section;
dlt_cnt = 0;
count = 1;

/*********************************************************************/
/*  Loop through the entry list and delete old spooled files         */
/*********************************************************************/
while (count <= space->Number_List_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 = (Qus_SPLA0100_t *)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);
/*********************************************************************/
/*  Convert the spooled file number to character.                    */
/*********************************************************************/
   memcpy (sp_spl_number,"      ",6);
   sprintf(tmp_spl_number,"%d",Rcv_Spl_Var->Splf_Number);
   memcpy(sp_spl_number,tmp_spl_number,strlen(tmp_spl_number));
/*********************************************************************/
/*  Delete the spooled file.                                         */
/*********************************************************************/
   CLDLT(sp_job_name,job_nmbr,usr_nm,
         job_nm,sp_spl_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 an ILE C program, specify the following:

```
CRTBNDC PGM(QGPL/DLTOLDSPLF) SRCFILE(QGPL/QCSRC)
```

CL Delete (CLDLT) program

The DLTOLDSPLF program, written in OPM RPG, OPM COBOL, or ILE C, 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('Spooled file ' *CAT &FILNAM *CAT        +
                         ' number ' *CAT &FILNUM *CAT ' job '    +
                         *CAT &JOBNUM *CAT  '/'                  +
                         *CAT &USRNAM *CAT '/' *CAT &JOBNAM *CAT +
                         ' deleted.')                            +
                    TOUSR(*REQUESTER) ENDPGM
```

To create the CL program, specify the following:

```
CRTCLPGM PGM(QGPL/CLDLT) SRCFILE(QGPL/QCLSRC)
```

Example: Diagnostic reporting

This ILE C program produces a diagnostic report of errors that occur when the Send Nonprogram Message (QMHSNDM) API is used to send a message to multiple message queues that do not exist.

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 QMHCHGEM API to handle the exception message. The QMHRCVPM API is used to receive the diagnostic messages. The program prints the exception message, the diagnostic messages, and the message help.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Diagnostic Report (DIAGRPT) program

```
/*****************************************************************/
/*                                                               */
/* MODULE NAME:  DIAGRPT - Diagnostic Report                     */
/* LANGUAGE:     ILE C                                           */
/*                                                               */
/* 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.  As a result, the generic CPF2469    */
/*               exception is returned indicating that one or more */
/*               diagnostic messages were returned identifying the */
/*               error(s) on the send operation.                 */
/*                                                               */
/*               The program looks for and handles the CPF2469   */
/*               exception.  It then receives and prints out the */
/*               exception and the previous diagnostics.         */
/*                                                               */
/* 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 <except.h>
#include <qmhchgem.h>                     /* From QSYSINC/H         */
#include <qmhrcvpm.h>                     /* From QSYSINC/H         */
#include <qmhsndm.h>                      /* From QSYSINC/H         */
#include <qusec.h>                        /* From QSYSINC/H         */

#define  DIAG_TYPE "02"
#define  BUF_SIZE  80

/*********************************************************************/
/* Type definition for error code structure                          */
/*********************************************************************/
typedef struct error_code_struct
  {
     Qus_EC_t   ec_fields;
     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
  {
   Qmh_Rcvpm_RCVM0200_t   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.              */
/*********************************************************************/
static void excp_handler(_INTRPT_Hndlr_Parms_T *excp_info)
 {
   error_code_struct Error_Code;

   /* If the exception is CPF2469, increment the exception counter,  */
   /* and mark the exception as handled by the QMHCHGEM API         */

   if (strncmp(excp_info->Msg_Id,"CPF2469",7) == 0) {
      memcpy(received,(excp_info->Msg_Id),7);
      exception_count++;
      QMHCHGEM(&(excp_info->Target), 0,
               (char *)(&(excp_info->Msg_Ref_Key)),
               "*HANDLE   ", "", 0, &Error_Code);
   }
 }


/*********************************************************************/
```

```
/* 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 and 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->",20);
  strncpy(buf+21,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 and 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 string  */

  PrintData("Message ID->",Msg->F.Message_Id,7);
  /* Convert Message Type to a character string to be printed out   */
  if (memcmp(Msg->F.Message_Type,"02",2)==0)
    strncpy(CharType,"DIAGNOSTIC", 10);
  else if (memcmp(Msg->F.Message_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.Length_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.Length_Message_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.Length_Help_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;
 memcpy(ErrorCode.ec_fields.Exception_Id,"       ",7);
 ErrorCode.ec_fields.Bytes_Provided = 0;

 memcpy(MsgId,"       ",7);
 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);


 /* Enable the exception handler around the call to QMHSNDM         */
 #pragma exception_handler(excp_handler, 0, 0, _C2_MH_ESCAPE)

 /* Send the message to the list of message queues.                 */
 QMHSNDM( MsgId,
         &MsgFile,
          MsgText,
          MsgTextLen,
          MsgType,
         &MsgQList,
          NumMsgQ,
         &RpyMsgQ,
         &MsgKey,
         &ErrorCode);

 /* Disable the exception handler                                   */
 #pragma disable_handler

 /* 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 and    */
   /* 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.ec_fields.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.ec_fields.Bytes_Available > 0)
     {
      PrintError("QMHRCVPM - Did not complete successfully",
                 ErrorCode.ec_fields.Exception_Id);
     }

   /* 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.Message_Id,received,7) != 0)
     {
      PrintError("QMHRCVPM - Wrong exception received",
                 MsgInfo.F.Message_Id);
     }


   /* The exception message was received successfully.             */
   /* Print the message data and text for the exception message.   */
   PrintMessage(&MsgInfo);

   /* If the message was the generic CPF2469, there are one 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.Message_Id,"CPF2469",7) == 0)
      {
       memcpy(MsgType,"*PRV      ",10);
       memcpy(MsgKey,MsgInfo.F.Message_Key,4);
       morediag = 1;

       while(morediag)
         {
          /* Receive the previous diagnostic */
          QMHRCVPM(&MsgInfo,
                   MsgInfoLen,
                   Format,
                   PgmMsgQ,
                   PgmCount,
                   MsgType,
                   MsgKey,
                   WaitTime,
                   MsgAction,
                   &ErrorCode);

          /* Test for error on the receive */
          if (ErrorCode.ec_fields.Bytes_Available > 0)
            {
             PrintError("QMHRCVPM - Did not complete successfully",
                        ErrorCode.ec_fields.Exception_Id);
            }

          /* 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.Message_Type,DIAG_TYPE,2) != 0) )
            {
             morediag = 0;
            }
          else                /* A diagnostic was received          */
            {
             /* Print the message data and text for the diagnostic  */
             /* message                                             */
             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.Message_Key,7);
            }

         }                        /* End of while morediag = 1      */

      }                           /* End of if CPF2469 received     */

    /* Write trailer */
    memset(buf,' ',BUF_SIZE);
    strncpy(buf,"-                        END OF DIAGNOSTIC REPORT",48);
    fwrite(buf,1,BUF_SIZE,prtf);

    /* Close the print file */
    fclose(prtf);

   }                              /* End of if error on send        */

}                                 /* End mainline                   */
```

Printed diagnostic report

The DIAGRPT program produces a report like this:

```
Message ID->CPF2469
Message Type->ESCAPE
Message text received->Error occurred when sending message.
Message help text received->Recovery  . . . :   See messages
          previously listed for a description of the error.
          Correct the error, and then try the
          command again.

Message ID->CPF2403
Message Type->DIAGNOSTIC
Message data received->PEANUTS   *LIBL
Message text received->Message queue PEANUTS 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->SNOOPY    *LIBL
Message text received->Message queue SNOOPY 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.
                     End of Diagnostic Report
```

Example: Generating and sending an alert

This ILE RPG program uses alert APIs. First, it calls the Generate Alert (QALGENA) API to generate an alert without sending a message to the QSYSOPR or QHST message queue. Then, it uses the Send Alert (QALSNDA) API to send the alert to the alert manager.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
     H
     D*************************************************************
     D*************************************************************
     D*
     D*  Program Name: ALERTS
     D*
     D*  Programming Language: ILE RPG
     D*
     D*  Description:  This program uses alert APIs.  First, it
     D*                calls the Generate Alert (QALGENA) API to
     D*                generate an alert without sending a message
     D*                to QSYSOPR or QHST message queue.  Then it
```

```
D*                uses the Send Alert (QALSNDA) API to send
D*                the alert to the alert manager.
D*
D*  Header Files Included: QUSEC - Error Code Parameter
D*
D*****************************************************************
D*****************************************************************
D*
D* Error Code parameter include
D*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
D*
D* Miscellaneous data structure
D*
DRCVVAR           S            512
DRCVLEN           S              9B 0 INZ(%SIZE(RCVVAR))
DALERT_SIZE       S              9B 0
DMSG_FILE         S             20    INZ('QCPFMSG   QSYS')
DMSG_ID           S              7    INZ('CPA2601')
DMSG_DATA         S            100
DMSG_SIZE         S              9B 0 INZ(0)
DALERT_TYPE       S              1    INZ('L')
DORIGIN           S             10    INZ('ALERTS')
C*
C* Beginning of mainline
C*
C* Set error handling
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Start by generating an alert for a specific message
C*
C                   CALL      'QALGENA'
C                   PARM                    RCVVAR
C                   PARM                    RCVLEN
C                   PARM                    ALERT_SIZE
C                   PARM                    MSG_FILE
C                   PARM                    MSG_ID
C                   PARM                    MSG_DATA
C                   PARM                    MSG_SIZE
C                   PARM                    QUSEC
C*
C* If no error reported, send the generated alert
C*
C     QUSBAVL       IFEQ      0
C                   CALL      'QALSNDA'
C                   PARM                    RCVVAR
C                   PARM                    ALERT_SIZE
C                   PARM                    ALERT_TYPE
C                   PARM                    ORIGIN
C                   PARM                    QUSEC
C*
C* If error on send, then display the error message
C*
C     QUSBAVL       IFNE      0
C     QUSEI         DSPLY
C                   END
C*
C* If error on generation, then display the error message
C*
C                   ELSE
C     QUSEI         DSPLY
C                   END
C*
C                   EVAL      *INLR = '1'
```

```
     C                   RETURN
     C*
     C* End of MAINLINE
     C*
```

Example: Listing directories

This ILE C program creates a report listing the contents of a directory.

You should call this program with only one parameter, the parameter that represents the directory you want to list.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/* FUNCTION:  This program lists a directory to a spooled file.   */
/*                                                                   */
/* LANGUAGE:  ILE C                                    */
/*                                                                   */
/*                                                                   */
/* APIs USED:  QHFOPNDR, QHFRDDR, QHFCLODR, QHFLSTFS, QUSCRTUS,   */
/*             QUSRTVUS                                              */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

/*********************************************************************/
/* INCLUDE FILES                                                     */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qhfopndr.h>
#include <qhfrddr.h>
#include <qhfclodr.h>
#include <qhflstfs.h>
#include <quscrtus.h>
#include <qusrtvus.h>
#include <qusec.h>

/*********************************************************************/
/* STRUCTURE AND VARIABLE DECLARATIONS                               */
/*********************************************************************/

/*********************************************************************/
/* Parameters for QHFOPNDR                                           */
/*********************************************************************/
char   dir_handle[16];           /* Directory handle              */
int    namelen;                  /* Length of path name           */
char   openinfo[6];              /* Open information              */

typedef struct {
        Qhf_Attr_Selec_Tbl_t fixed;
        int                  offset2;
        int                  offset3;
        int                  att_len1;
        char                 att_name1[8];
        int                  att_len2;
        char                 att_name2[8];
        int                  att_len3;
        char                 att_name3[8];
} selection_struct;

selection_struct select;
```

```
int              selectionlen;

/*****************************************************************/
/* Error Code Structure                                          */
/*                                                               */
/* This shows how the user can define the variable length portion */
/* of error code for the exception data.                         */
/*                                                               */
/*****************************************************************/
typedef struct {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
        } error_code_t;

error_code_t  error_code;

/*****************************************************************/
/* Parameters for QHFRDDR                                        */
/*****************************************************************/
/* The directory handle is the same as for QHFOPNDR              */

typedef struct {
        Qhf_Data_Buffer_t   fixed;
        int                 num_att;
        int                 offsets[4];
        char                attinfo[276];
} read_buffer;

read_buffer buffer;
int    result_count;
int    bytes_returned;

/*****************************************************************/
/* Parameters for QHFCLODR                                       */
/*****************************************************************/
/* No additional ones need to be declared                        */

/*****************************************************************/
/* Parameters for QUSCRTUS                                       */
/*****************************************************************/
int    size;
char   text[50];

/*****************************************************************/
/* Parameters for QHFLSTFS                                       */
/*****************************************************************/
/* No additional ones need to be declared                        */

/*****************************************************************/
/* Parameters for QUSRTVUS                                       */
/*****************************************************************/
int    startpos;
int    len;
char   charbin4[4];
char   FSname[10];

/*****************************************************************/
/* Other declarations                                            */
/*****************************************************************/
int    entrypos;
int    numentries;
int    entrylen;
char   *att;
char   name[100];
char   attname[30];
char   attval[30];
int    attnamelen;
```

```
int    attvallen;
char   newname[30];
int    filesize;
char   fileatt[10];

typedef struct {
        char     century;
        char     year[2];
        char     month[2];
        char     day[2];
        char     hour[2];
        char     minute[2];
        char     second[2];
} charval;

charval chartime;

int    bytes_used;
int    i;

main(int argc, char *argv[])
{
  char    write_string[100];
  FILE    *stream;

  error_code.ec_fields.Bytes_Provided = 0;

  /*****************************************************************/
  /* Make sure we received the correct number of parameters. The  */
  /* argc parameter will contain the number of parameters that    */
  /* was passed to this program. This number also includes the    */
  /* program itself, so we need to evaluate argc-1.               */
  /*****************************************************************/

  if (((argc - 1) < 1) || ((argc - 1 > 1)))
  /*****************************************************************/
  /* We did not receive all of the required parameters, or        */
  /* received too many. Exit from the program.                    */
  /*****************************************************************/
  {
    exit(1);
  }

  /*****************************************************************/
  /* Open QPRINT file so that data can be written to it.  If the  */
  /* file cannot be opened, print a message and exit.             */
  /*****************************************************************/
  if((stream = fopen("QPRINT", "wb")) == NULL)
  {
    printf("File could not be opened\n");
    exit(1);
  }

  memset(name, ' ', 100);
  memcpy(name, argv[1], 100);
  if(!memcmp(name, " ", 1))
  {
  memcpy(name,"ROOT",4);
  fprintf(stream,"Directory listing for path %.100s\n", name);
  size = 1;
  memcpy(text, "temporary user space used by program DIR          ",
          50);
  /*****************************************************************/
  /* Create the user space for QHFLSTFS to use.                   */
  /*****************************************************************/
  QUSCRTUS("FSLST     QTEMP     ", "TEMPSPACE ", size, " ",
           "*USE      ", text, "*YES      ", &error_code);
```

```
/*************************************************************/
/* List the file systems into that space.                    */
/*************************************************************/
QHFLSTFS("FSLST     QTEMP     ", "HFSL0100", &error_code);

/*************************************************************/
/* Get the starting point for the file system entries.       */
/*************************************************************/
startpos = 125;
len = 4;
QUSRTVUS("FSLST     QTEMP     ", startpos, len, charbin4,
         &error_code);

entrypos = *(int *)charbin4;

/*************************************************************/
/* Get the number of entries in the user space.              */
/*************************************************************/
startpos = 133;
len = 4;

QUSRTVUS("FSLST     QTEMP     ", startpos, len, charbin4,
         &error_code);

numentries = *(int *)charbin4;

/*************************************************************/
/* Find the length of the entries.                           */
/*************************************************************/
startpos = 137;
len = 4;

QUSRTVUS("FSLST     QTEMP     ", startpos, len, charbin4,
         &error_code);

entrylen = *(int *)charbin4;

/*************************************************************/
/* Loop through the entries and get the names of the file    */
/* systems.                                                  */
/*************************************************************/
for(i=0;i<numentries;++i)
{
   startpos = entrypos + 1;
   len = 10;
   QUSRTVUS("FSLST     QTEMP     ", startpos, len, FSname,
            &error_code);
   /*********************************************************/
   /* List the names into the spooled file.                 */
   /*********************************************************/
   sprintf(write_string," %.10s <DIR>", FSname);
   fprintf(stream, write_string);
   entrypos = entrypos + entrylen;
}

}
else
{
fprintf(stream,"Directory listing for path %.100s\n", name);
/*************************************************************/
/* Build the attribute selection table for QHFOPNDR.         */
/*************************************************************/
select.fixed.Number_Attributes = 3;
select.fixed.Offset_First_Attr = 16;
select.offset2 = 28;
select.offset3 = 40;
```

```
select.att_len1 = 8;
memcpy(select.att_name1, "QFILSIZE", 8);
select.att_len2 = 8;
memcpy(select.att_name2, "QCRTDTTM", 8);
select.att_len3 = 8;
memcpy(select.att_name3, "QFILATTR", 8);
selectionlen = 52;
memcpy(openinfo, "10    ", 6);

/*************************************************************/
/* Find the length of the directory name.                    */
/*************************************************************/
for(i=0;i<100;i++)
{
  if((name[i] == ' ') || (name[i] == '\x00'))
    break;
}
namelen = i;

/*************************************************************/
/* Open the directory.                                       */
/*************************************************************/
QHFOPNDR(dir_handle, name, namelen, openinfo, &select, selectionlen,
         &error_code);

/*************************************************************/
/* Read one entry from the directory.                        */
/*************************************************************/
QHFRDDR(dir_handle, &buffer, 300, 1, &result_count, &bytes_returned,
        &error_code);

while(result_count > 0)
{
  memcpy(attname,"                               ",30);
  memcpy(attval,"                              ",30);
  att = buffer.attinfo;
  bytes_used = 20;

  /***********************************************************/
  /* Loop for the number of attributes in the entry.         */
  /***********************************************************/
  for(i=0;i<buffer.num_att;i++)
  {
    memcpy(charbin4, att, 4);
    attnamelen = *(int *)charbin4;
    att += 4;
    bytes_used += 4;
    memcpy(charbin4, att, 4);
    attvallen = *(int *)charbin4;
    att += 8;
    bytes_used += 8;
    memcpy(attname, att, attnamelen);
    att += attnamelen;
    bytes_used += attnamelen;
    memcpy(attval, att, attvallen);
    att += attvallen;
    bytes_used += attvallen;

    /*********************************************************/
    /* Update att so that its first character is the first   */
    /* character of the next attribute entry.                */
    /*********************************************************/
    if ((bytes_used == buffer.offsets[i+1]) &&
        ((i+1) == buffer.num_att))
      att += (buffer.offsets[i] - bytes_used);

    /*********************************************************/
```

```
    /* If the attribute is QNAME, then set newname.             */
    /************************************************************/
    if(!memcmp(attname, "QNAME", 5))
    {
      memset(newname, ' ', 12);
      memcpy(newname, attval, attvallen);
    }

    /************************************************************/
    /* If the attribute is QFILSIZE, then set filesize.         */
    /************************************************************/
    else if(!memcmp(attname, "QFILSIZE", 8))
    {
      memcpy(charbin4, attval, 4);
      filesize = *(int *)charbin4;
    }
    /************************************************************/
    /* If it was QCRTDTTM, then set the time.                   */
    /************************************************************/
    else if(!memcmp(attname, "QCRTDTTM", 8))
      memcpy(&chartime, attval, 13);
    /************************************************************/
    /* Else the attribute was QFILATTR, so set fileatt.         */
    /************************************************************/
    else
      memcpy(fileatt, attval, 10);
  }

  /************************************************************/
  /* If the entry was a directory, list its name and <DIR>.     */
  /************************************************************/
  if(fileatt[3] == '1')
  {
    sprintf(write_string," %s <DIR>", newname);
    fprintf(stream, write_string);
  }
  /************************************************************/
  /* If the entry is not a hidden file, list its name and size. */
  /************************************************************/
  else if(fileatt[1] == '0')
  {
    sprintf(write_string," %s %d", newname, filesize);
    fprintf(stream, write_string);
  }
  /************************************************************/
  /* If the entry is not a hidden file or directory, list its   */
  /* date of creation.                                        */
  /************************************************************/
  if(fileatt[1] == '0')
  {
    sprintf(write_string," %.2s-%.2s-%.2s", chartime.month,
            chartime.day, chartime.year);
    fprintf(stream, write_string);
    sprintf(write_string," %.2s:%.2s:%.2s\n", chartime.hour,
            chartime.minute, chartime.second);
    fprintf(stream, write_string);
  }

  QHFRDDR(dir_handle, &buffer, 200, 1, &result_count, &bytes_returned,
          &error_code);

} /* while */

} /* else */

/************************************************************/
/* Close the directory.                                     */
```

```
  /*************************************************************/
  QHFCLODR(dir_handle, &error_code);

  fclose(stream);

} /* main */
```

Example: Listing subdirectories

This ILE C program creates a report listing the subdirectories of a directory.

You should call this program with only one parameter, the parameter that represents the directory you want to list.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/* FUNCTION:                          */
/*                                                                   */
/* LANGUAGE:  ILE C                                  */
/*                                                                   */
/*                                                                   */
/* APIs USED:  QHFOPNDR, QHFRDDR, QHFCLODR                           */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

/*********************************************************************/
/* INCLUDE FILES                                                     */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qhfopndr.h>
#include <qhfrddr.h>
#include <qhfclodr.h>
#include <qusec.h>

char    write_string[100];
FILE    *stream;

void print_subdir(char name[100], int numtabs)
{

/*********************************************************************/
/* Parameters for QHFOPNDR                                           */
/*********************************************************************/
char   dir_handle[16];           /* Directory handle              */
int    namelen;                  /* Length of path name           */
char   openinfo[6];              /* Open information              */

typedef struct {
        Qhf_Attr_Selec_Tbl_t fixed;
        int                 att_len;
        char                att_name[8];
} selection_struct;

selection_struct select;
int              selectionlen;

/*********************************************************************/
/* Error Code Structure                                              */
/*                                                                   */
/* This shows how the user can define the variable length         */
```

```
/* portion of error code for the exception data.                 */
/*                                                               */
/*****************************************************************/
typedef struct {
     Qus_EC_t   ec_fields;
     char       Exception_Data[100];
        } error_code_t;

error_code_t  error_code;

/*****************************************************************/
/* Parameters for QHFRDDR                                        */
/*****************************************************************/
/* The directory handle is the same as for QHFOPNDR              */

typedef struct {
        Qhf_Data_Buffer_t   fixed;
        int                 num_att;
        int                 offsets[2];
        char                attinfo[180];
} read_buffer;

read_buffer buffer;
int    result_count;
int    bytes_returned;

/*****************************************************************/
/* Parameters for QHFCLODR                                       */
/*****************************************************************/
/* No additional ones need to be declared                        */

/*****************************************************************/
/* Other declarations                                            */
/*****************************************************************/
char   *att;
char   attname[30];
char   attval[30];
int    attnamelen;
int    attvallen;
char   newname[30];
int    newnamelen;
int    filesize;
char   fileatt[10];
char   tab[5];
int    bytes_used;
int    i, j;
char   charbin4[4];
char   tempname[100];

  error_code.ec_fields.Bytes_Provided = 0;

  /***************************************************************/
  /* Build the attribute selection table for QHFOPNDR.           */
  /***************************************************************/
  select.fixed.Number_Attributes = 1;
  select.fixed.Offset_First_Attr = 8;
  select.att_len = 8;
  memcpy(select.att_name, "QFILATTR", 8);
  selectionlen = 20;
  memcpy(openinfo, "10    ", 6);
  memcpy(tab,"     ", 5);

  /***************************************************************/
  /* Find the length of the directory name.                      */
  /***************************************************************/
  for(i=0;i<100;i++)
  {
```

```
  if((name[i] == ' ') || (name[i] == '\x00'))
    break;
}
namelen = i;

/*************************************************************/
/* Open the directory.                                       */
/*************************************************************/
QHFOPNDR(dir_handle, name, namelen, openinfo, &select, selectionlen,
         &error_code);

/*************************************************************/
/* Read one entry from the directory.                        */
/*************************************************************/
QHFRDDR(dir_handle, &buffer, 200, 1, &result_count, &bytes_returned,
        &error_code);

fprintf(stream,"\n");
for(i=0;i<numtabs;i++)
  fprintf(stream, tab);
fprintf(stream, name);

while(result_count > 0)
{
  memcpy(attname,"                               ",30);
  memcpy(attval,"                               ",30);
  att = buffer.attinfo;
  bytes_used = 12;

  /*************************************************************/
  /* Loop for the number of attributes in the entry.           */
  /*************************************************************/
  for(i=0;i<buffer.num_att;i++)
  {
    memcpy(charbin4, att, 4);
    attnamelen = *(int *)charbin4;
    att += 4;
    bytes_used += 4;
    memcpy(charbin4, att, 4);
    attvallen = *(int *)charbin4;
    att += 8;
    bytes_used += 8;
    memcpy(attname, att, attnamelen);
    att += attnamelen;
    bytes_used += attnamelen;
    memcpy(attval, att, attvallen);
    att += attvallen;
    bytes_used += attvallen;

    /*********************************************************/
    /* Update att so that its first character is the first   */
    /* character of the next attribute entry.                */
    /*********************************************************/
    if ((bytes_used == buffer.offsets[i+1]) &&
        ((i+1) == buffer.num_att))
      att += (buffer.offsets[i] - bytes_used);

    /*********************************************************/
    /* If the attribute is QNAME, then set newname and       */
    /* newnamelen just in case the entry is a directory.     */
    /*********************************************************/
    if(!memcmp(attname, "QNAME", 5))
    {
      memcpy(newname, attval, attvallen);
      newnamelen = attvallen;
    }
```

```
      /*************************************************************/
      /* Else the attribute was QFILATTR, so set fileatt.          */
      /*************************************************************/
      else
        memcpy(fileatt, attval, 10);
    }

    /*************************************************************/
    /* If the entry was a directory, construct new path name and  */
    /* print_subdir to print the subdirectory.                    */
    /*************************************************************/
    if(fileatt[3] == '1')
    {
      memcpy(tempname, name, 100);
      strcat(name, "/");
      strcat(name, newname);
      memcpy(newname, name, namelen + newnamelen + 1);
      print_subdir(newname, numtabs + 1);
      memcpy(name, tempname, 100);
    }

    QHFRDDR(dir_handle, &buffer, 200, 1, &result_count, &bytes_returned,
            &error_code);

  } /* while */

  /*****************************************************************/
  /* Close the directory.                                          */
  /*****************************************************************/
  QHFCLODR(dir_handle, &error_code);

}/* print_subdir */


main(int argc, char *argv[])
{
  char    dir_name[100];

  /*****************************************************************/
  /* Make sure we received the correct number of parameters. The  */
  /* argc parameter will contain the number of parameters that    */
  /* was passed to this program. This number also includes the    */
  /* program itself, so we need to evaluate argc-1.               */
  /*****************************************************************/

  if (((argc - 1) < 1) || ((argc - 1 > 1)))
  /*****************************************************************/
  /* We did not receive all of the required parameters, or        */
  /* received too many. Exit from the program.                    */
  /*****************************************************************/
  {
    exit(1);
  }

  /*****************************************************************/
  /* Open QPRINT file so that data can be written to it.  If the  */
  /* file cannot be opened, print a message and exit.             */
  /*****************************************************************/
  if((stream = fopen("QPRINT", "wb")) == NULL)
  {
    printf("File could not be opened\n");
    exit(1);
  }

  memset(dir_name, ' ', 100);
  memcpy(dir_name, argv[1], 100);
  if(!memcmp(dir_name, " ", 1))
```

```
  {
  fprintf(stream,"No directory specified");
  }
  else
  {
  fprintf(stream,"Directory substructure starting at %.100s", dir_name);
  print_subdir(dir_name, 0);
  }

  fclose(stream);

} /* main */
```

Example: Saving to multiple devices

This ILE C program saves a large library using more than one device at the same time.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*****************************************************************/
/*  PROGRAM:  SaveBigLib                                         */
/*                                                               */
/*  LANGUAGE: ILE C                                              */
/*                                                               */
/*  DESCRIPTION:  This is an example program for the use of      */
/*                a media definition in a save operation.        */
/*                It saves library BIGLIB in parallel format to  */
/*                two media files, using tape media library      */
/*                TAPMLB01.                                      */
/*                                                               */
/*                The flow of this program is as follows:        */
/*                (1) Build media definition input.              */
/*                (2) Create a media definition using            */
/*                    QsrCreateMediaDefinition.                  */
/*                (3) Save library BIGLIB using the media        */
/*                    definition.                                */
/*                                                               */
/*  APIs USED:  QsrCreateMediaDefinition, QCMDEXC                */
/*                                                               */
/*****************************************************************/

#include <qcmdexc.h>
#include <qsrlib01.h>
#include <qusec.h>
#include <string.h>

/*****************************************************************/
/* Variables for QsrCreateMediaDefinition                        */
/*****************************************************************/
char                      Data_Buffer[1000];
Qsr_TAPE0100_t            *Input_Data;
Qsr_TAPE0100_Device_t     *Device;
Qsr_TAPE0100_File_t       *Media_File;
char                      *Next_Free;
char                      *Volid;
Qus_EC_t                  Err_Code;
int                       Data_Length;
char                      Text[50];

/*****************************************************************/
/* Variables for QCMDEXC                                         */
/*****************************************************************/
char          Cmd_String[100];
decimal(15,5) Cmd_Length;

/*****************************************************************/
```

```
/* Start of main()                                                   */
/*******************************************************************/

int main (int argc, char *argv[]) {

/*******************************************************************/
/*  Specify input data for QsrCreateMediaDefinition.               */
/*******************************************************************/
/*-----------------------------------------------------------------*/
/*  Build general media definition input data.                     */
/*  Use one device with two parallel device resources.             */
/*-----------------------------------------------------------------*/
memset(Data_Buffer,0,sizeof(Data_Buffer));
Input_Data = (Qsr_TAPE0100_t*)Data_Buffer;
Next_Free = (char*)(Input_Data + 1);
Input_Data->Maximum_Resources    = 2;
Input_Data->Minimum_Resources    = 2;
Input_Data->Offset_First_Device  = Next_Free - Data_Buffer;
Input_Data->Device_Count         = 1;

/*-----------------------------------------------------------------*/
/*  Build input data for the first device.                         */
/*  Use device TAPMLB01 with two media files.                      */
/*-----------------------------------------------------------------*/
Device = (Qsr_TAPE0100_Device_t*)Next_Free;
Next_Free = (char*)(Device + 1);
memcpy(Device->Device_Name,"TAPMLB01  ",10);
Device->Offset_First_File = Next_Free - Data_Buffer;
Device->File_Count        = 2;

/*-----------------------------------------------------------------*/
/*  Build input data for the first media file for device TAPMLB01. */
/*  Use the default sequence number, and volumes VOL11 and VOL12.  */
/*-----------------------------------------------------------------*/
Media_File = (Qsr_TAPE0100_File_t*)Next_Free;
Next_Free = (char*)(Media_File + 1);
Media_File->Sequence_Number        = 0;
Media_File->Offset_First_Volume_Id = Next_Free - Data_Buffer;
Media_File->Volume_Id_Count        = 2;
Media_File->Volume_Id_Length       = 6;
Media_File->Starting_Volume        = 1;
Data_Length = Media_File->Volume_Id_Count
            * Media_File->Volume_Id_Length;
Volid = Next_Free;
memcpy(Volid,"VOL11 VOL12 ",Data_Length);
if (Data_Length % 4)                       /* Ensure that Next_Free */
  Data_Length += (4 - (Data_Length % 4));  /* is incremented by a   */
Next_Free += Data_Length;                  /* multiple of 4.        */
Media_File->Offset_Next_File = Next_Free - Data_Buffer;

/*-----------------------------------------------------------------*/
/*  Build input data for the second media file for device TAPMLB01. */
/*  Use the default sequence number, and volumes VOL21 and VOL22.   */
/*-----------------------------------------------------------------*/
Media_File = (Qsr_TAPE0100_File_t*)Next_Free;
Next_Free = (char*)(Media_File + 1);
Media_File->Sequence_Number        = 0;
Media_File->Offset_First_Volume_Id = Next_Free - Data_Buffer;
Media_File->Volume_Id_Count        = 2;
Media_File->Volume_Id_Length       = 6;
Media_File->Starting_Volume        = 1;
Data_Length = Media_File->Volume_Id_Count
            * Media_File->Volume_Id_Length;
Volid = Next_Free;
memcpy(Volid,"VOL21 VOL22 ",Data_Length);
if (Data_Length % 4)                       /* Ensure that Next_Free */
  Data_Length += (4 - (Data_Length % 4));  /* is incremented by a   */
```

```
Next_Free += Data_Length;                  /* multiple of 4.        */

/*********************************************************************/
/*  Create the media definition.                                     */
/*********************************************************************/
Data_Length = Next_Free - Data_Buffer;
memset(Text,' ',sizeof(Text));
memcpy(Text,"Save BIGLIB",11);
QsrCreateMediaDefinition(
                  "SAVEBIGLIBQTEMP     ",  /* Media definition      */
                                           /* name, library     */
                  Data_Buffer,             /* Input data            */
                  Data_Length,             /* Length of data        */
                  "TAPE0100",              /* Format name           */
                  "*USE      ",            /* Public authority      */
                  Text,                    /* Text description      */
                  '1',                     /* Replace if it exists  */
                  &Err_Code);              /* Error code            */
/*********************************************************************/
/*  Save library BIGLIB using the media definition.                  */
/*********************************************************************/
strcpy(Cmd_String,
       "SAVLIB LIB(BIGLIB) DEV(*MEDDFN) MEDDFN(QTEMP/SAVEBIGLIB)");
Cmd_Length = strlen(Cmd_String);
QCMDEXC(Cmd_String,Cmd_Length);

return 0;
}
```

Example: Saving and restoring system-level environment variables

These ILE C programs show how to save the current set of system-level environment variables and restore them later.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Saving system-level environment variables

This program saves the system-level environment variables and the associated CCSIDs in a file for restoring later.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program.

Call this program with one parameter (the file to store the variable list and the CCSIDs).

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  Save the system-level environment variable list        */
/*            and the CCSIDs in a file                               */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: Qp0zGetAllSysEnv()                                     */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <stdlib.h>
```

```
#include <errno.h>
#include <qp0z1170.h>


int main(int argc, char *argv[])
{

    int fd, bw, rc;
    int listBufSize, ccsidBufSize, *ccsidBuf;
    char *listBuf;
    int numvar, sl, sc;

    if(argc != 2)
    {
      printf("Usage: call %s <filename>\n",argv[0]);
      printf("Example: call %s '/tmp/sev'\n",argv[0]);
      return -1;
    }

    sl = listBufSize = 1000;
    sc = ccsidBufSize = 1000;
    listBuf = (char *)malloc(listBufSize);
    ccsidBuf = (int *)malloc(ccsidBufSize);

    /* Create a file of specified name */
    /* If it exists, it is cleared out */
    /* Opened for writing              */
    fd = open(argv[1], O_CREAT | O_WRONLY | O_TRUNC, S_IRWXU);
    if(fd == -1)
    {
      printf("open() failed. errno = %d\n", errno);
      return -1;
    }


    rc = Qp0zGetAllSysEnv(listBuf, &listBufSize, ccsidBuf,
                          &ccsidBufSize, NULL);
    if(rc != 0)
    {
      /* If there are no variables to save, write a  */
      /* zero into the file and return success       */

      if(rc == ENOENT)
      {
        numvar = 0;
        bw = write(fd, &numvar, sizeof(int));
        close(fd);
        printf("No system-level environment variables to save");
        return 0;
      }

      if(rc != ENOSPC)
      {
        printf("Error using  Qp0zGetAllSysEnv(), errno = %d\n", rc);
        return -1;
      }

      /* rc = ENOSPC. size of buffer is not enough */
      /* change buffer size and try again */

      /* If listBuf is not large enough, */
      /* allocate more space             */
      if(listBufSize > sl)
      {
        listBuf = (char *)realloc(listBuf, listBufSize);
      }
```

```
  /* If ccsidBuf is too small, allocate */
  /* more space                         */
  if(ccsidBufSize > sc)
  {
    ccsidBuf = (int *)realloc(ccsidBuf, ccsidBufSize);
  }

  rc = Qp0zGetAllSysEnv(listBuf, &listBufSize,
                        ccsidBuf, &ccsidBufSize, NULL);
  if(rc != 0)
  {
    printf("Error using Qp0zGetAllSysEnv(), errno = %d\n", rc);
    return -1;
  }
}


/* Write the contents of the buffer into the file */
/* First write the total number of ccsid values   */
/* This is the total number of variables          */

numvar = ccsidBufSize/sizeof(int);

bw = write(fd, &numvar, sizeof(int));
if(bw == -1)
{
  printf("write() of total number of ccsids failed. errno = %d\n", errno);
  return -1;
}


/* Next write the ccsid values */

bw = write(fd, ccsidBuf, ccsidBufSize);
if(bw == -1)
{
  printf("write() of ccsid values failed. errno = %d\n", errno);
  return -1;
}

/* Now write the size (in bytes) of the listBuf */

bw = write(fd, &listBufSize, sizeof(int));
if(bw == -1)
{
  printf("write() of listBufSize failed. errno = %d\n", errno);
  return -1;
}

/* Finally write the listBuf containing the variable strings*/

bw = write(fd, listBuf, listBufSize);
if(bw == -1)
{
  printf("write() of listBuf failed. errno = %d\n", errno);
  return -1;
}

/* Close the file */
rc = close(fd);
if(rc != 0)
{
  printf("close() failed. errno = %d\n", errno);
  return -1;
}
```

```
    printf("System-level environment variables saved\n");
    return 0;
}
```

Restoring system-level environment variables

This program reads the system-level environment variable list from a file and then sets the system-level environment variables.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program.

Call this program with one parameter (the name of the file in which the system-level environment variables were stored).

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  Restore the system-level environment variable list  */
/*            and the associated CCSIDs stored in a file          */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: Qp0zPutSysEnv()                                        */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <qp0z1170.h>


int main(int argc, char *argv[])
{
    int fd, rc, br, i, numvar;
    int ccsidBufSize = 0, listBufSize = 0, *ccsidBuf;
    char *listBuf;


    if (argc != 2)
    {
      printf("Usage: call %s <filename>\n",argv[0]);
      printf("Example: call %s '/tmp/sev'\n",argv[0]);
      return -1;
    }

    /* Open the file specified */

    fd = open(argv[1], O_RDONLY);
    if(fd == -1)
    {
      printf("open() failed. errno = %d\n", errno);
      return -1;
    }

    /* Get the number of variables */
    br = read(fd, &numvar, sizeof(int));
    if(br == -1)
    {
      printf("read() failed. errno = %d\n", errno);
      return -1;
```

```
    }

    /* Could delete the existing system-level environment */
    /* variables and have only the restored values.       */
    /* If so desired, could call Qp0zDltSysEnv() to do so */

    /* If there aren't any elements in the file, skip the rest of */
    /* the reads and go to the end                                */

    if(numvar > 0)
    {
      ccsidBufSize = numvar*sizeof(int);
      ccsidBuf = (int *)malloc(ccsidBufSize);

      /* Read the ccsid values and put it in ccsidBuf */
      br = read(fd, ccsidBuf, ccsidBufSize);
      if(br == -1)
      {
        printf("read() failed. errno = %d\n", errno);
        return -1;
      }


      /* Read the size of the list buffer and put it in listBufSize */
      br = read(fd, &listBufSize, sizeof(int));
      if(br == -1)
      {
        printf("read() failed. errno = %d\n", errno);
        return -1;
      }

      listBuf = (char *)malloc(listBufSize);

      /* Finally read the strings themselves */
      br = read(fd, listBuf, listBufSize);
      if(br == -1)
      {
        printf("read() failed. errno = %d\n", errno);
        return -1;
      }
   }

   /* Walk through the buffer and get the   */
   /* name=value strings one by one         */
   /* Use Qp0zPutSysEnv() to set the values */

   for(i = 0; i < numvar; i++)
   {
     rc = Qp0zPutSysEnv(listBuf, ccsidBuf[i], NULL);
     if(rc != 0)
     {
       printf("Qp0zPutSysEnv() failed. rc=%d\n",rc);
       return -1;
     }

     listBuf += strlen(listBuf) + 1;
   }

   close(fd);
   printf("System-level environment variables restored\n");
   return 0;

}
```

Examples: Scanning string patterns

These examples use the Scan for String Pattern (QCLSCAN) API to retrieve all database records that contain a pattern.

Example 1

Assume that a 20-character database field contains only uppercase characters and the pattern 'ABC' is scanned for. The user program calls the QCLSCAN API for each database record that is read. The parameters follow.

Field name	Result
STRING	The 20-byte field to be scanned
STRLEN	20
STRPOS	1
PATTERN	'ABC'
PATLEN	3
TRANSLATE	'0'
TRIM	'0'
WILD	' '
RESULT	A value returned to your program

Some fields and the results of the scan follow:

```
Scan  String               Result  Comments
 1    ABCDEFGHIJKLMNOPQRST   001
 2    XXXXABCXXXXXXXXXXXXX   005
 3    abcXXXXXXXXXXXXXXXX   000    Translation not requested
 4    XXXABCXXXXXABCXXXXXX   004    First occurrence found; see note
 5    ABABABABBCACCBACBABA   000    Not found
 6    ABABABCABCABCABCABCA   005
```

Note: In scan 4, the string has two places where the pattern can be found. Because the STRPOS value is 1, the first value (position 004) is found. If the STRPOS value is 4, the result is still 004. If the STRPOS value is in a range of 5 through 12, the result is 012.

Example 2

Assume that a 25-character database field contains only uppercase characters. The user program prompts for the pattern that does not exceed 10 characters to be scanned for. The workstation user can enter 1 through 10 characters. The system trims trailing blanks from the pattern. The program calls the QCLSCAN API for each database record that is read. The parameters follow.

Field name	Result
STRING	The 25-byte field to be scanned
STRLEN	25
STRPOS	1
PATTERN	Varies
PATLEN	10
TRANSLATE	'0'
TRIM	'1'
WILD	' '

Field name	Result
RESULT	A value returned to your program

Some fields and the results of the scan follow:

```
Scan  String                    Pattern        Result  Comments
 1    ABCDEFGHIJKLMNOPQRSTUVWXY  'CDE        '   003
 2    ABCDEFGHIJKLMNOPQRSTUVWXY  'CDEFGH     '   003
 3    ABCDEFGHIJKLMNOPQRSTUVWXY  'CDEFGHIJKL '   003
 4    XXXXABCXXXXXXXXXXXXXXXXXX  'ABCD       '   000    Not found
 5    abcXXXXXXXXXXXXXXXXXXXXX  'ABC        '   000    Not translated
 6    ABCXXXXXABC EXXXXXXXXXXXX  'ABC E      '   009
 7    XXXABCXXXXXABCXXXXXXXXXXX  'ABC        '   004    See note
```

Note: In scan 7, the string has two places where the pattern can be found. Because the STRPOS value is 1, only the first value (position 004) is found. If the STRPOS value is 4, the result is still 004. If the STRPOS value is in a range of 5 through 12, the result is 012.

Example 3

Assume that a 25-character database field contains either uppercase or lowercase characters. The user program prompts for the pattern that does not exceed 5 characters to be scanned for. The workstation user can enter 1 through 5 characters. The system trims trailing blanks from the pattern. If the user enters an asterisk (*) in the pattern, the asterisk is handled as a wild character. The program calls the QCLSCAN API for each database record that is read. The parameters follow.

Field name	Result
STRING	The 25-byte field to be scanned
STRLEN	25
STRPOS	1
PATTERN	Varies
PATLEN	5
TRANSLATE	'1' (See note 1)
TRIM	'1'
WILD	'*'
RESULT	A value returned to your program

Some fields and the results of the scan follow:

```
Scan  String                    Pattern      Result  Comments
 1    ABCDEFGHIJKLMNOPQRSTUVWXY  'CDE      '   003
 2    ABCDEFGHIJKLMNOPQRSTUVWXY  'C*E      '   003
 3    abcdefghijklmnopqrstuvwxy  'C***G    '   003    See note 1
 4    abcdefghijklmnopqrstuvwxy  'ABCD     '   001
 5    abcXXXXXXXXXXXXXXXXXXXXX  'C*E      '   000    Not found
 6    XXXAbcXXXXXabcXXXXXXXXXXX  'ABC      '   004    See note 2
 7    ABCDEFGHIJKLMNOPQRSTUVWXY  '*BC      '  -003    See note 3
 8    ABCDEFGHIJKLMNOPQRSTUVWXY  '         '  -004    See note 4
```

Notes:

1. When field translation is specified (the TRANSLATE parameter is specified as '1'), the string is translated to uppercase characters before scanning occurs; the data in the string is not changed.
2. In scan 6, the string has two places where the pattern can be found. Because the STRPOS value is 1, the first value (position 004) is found.

3. In scan 7, the wild character (*) is the first character in the trimmed pattern. Wild characters cannot be the first character in a pattern.
4. In scan 8, the trimmed pattern is blank.

Example: Using a COBOL/400 program to call APIs

This COBOL/400 program creates a pending run unit and sets an error handler for the pending run unit.

The program uses the example error handler in "Error handler for the example COBOL/400 program" on page 384.

Notes:

- The error-handling program, ACERRF24 (shown in "Error handler for the example COBOL/400 program" on page 384), must exist in the UTCBL library.
- By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ACF24.
*************************************************************
*************************************************************
*
* FUNCTION:  SHOWS HOW TO CALL THE VARIOUS APIs, WHILE
*            TESTING THAT THEY WORK PROPERLY.
*
* LANGUAGE:  COBOL
*
* APIs USED: QLRRTVCE, QLRCHGCM, QLRSETCE
*
*************************************************************
*************************************************************
 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 the example COBOL/400 program

This example error handler works with "Example: Using a COBOL/400 program to call APIs" on page 382.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ACERRF24.
*************************************************************
*************************************************************
*
* FUNCTION:  Error handler for preceding example COBOL program
*
* LANGUAGE:  COBOL
*
* APIs USED: None
*
*************************************************************
*************************************************************
 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.
```

Examples: Using the Control Device (QTACTLDV) API

These programs show how to call the Control Device (QTACTLDV) API to issue a diagnostic command to a tape device and to display the firmware level of the tape device.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Example 1: Sending a diagnostic command to a tape device

```
/* Usage example 1 for QTACTLDV API                                  */
/*                                                                   */
/*                                                                   */

 #include <string.h>
 #include <stdio.h>
 #include <qtactldv.h>
 #include <qusec.h>

 /*********************************************************************/
 /*  Typedef structure for QTACTLDV                                   */
 /*********************************************************************/
 typedef struct {                          /* QTACTLDV command data  */
   Qta_CTLD0100_t  data;                   /*   command data         */
   char        cmd_str[6];                 /*   command string       */
 } cmd_struct;

 /*********************************************************************/
 /*  Typedef structure for Error code                                 */
 /*********************************************************************/
 typedef struct {                          /* Error code structure   */
   Qus_EC_t    Edata;                      /*   Error code data      */
                                           /* CPF67C8 excp data      */
   char        dev_nam[10];                /*   Device name          */
   char        reason_cd[3];               /*   Reason code          */
   char        resv1[3];                   /*   Reserved             */
 } EC_struct;

 /*********************************************************************/
 /*  Constants                                                        */
 /*********************************************************************/

 #define  SNDRSNS "\x03\x00\x00\x00\x12\x00"  /* Request sense       */
                                              /* command string      */
 #define  SNDDIAG "\x1D\x04\x00\x00\x00\x00"  /* Send diagnostic     */
                                              /* command string      */


 main(int argc,char *argv[])
 {
 /*********************************************************************/
 /*                                                                   */
 /*  START OF MAINLINE                                                */
 /*                                                                   */
 /*********************************************************************/

 /*********************************************************************/
 /*  Variables for QTACTLDV                                           */
 /*********************************************************************/
  char                 device[10];        /* device name            */
  char                 send_buff[256];    /* send buffer            */
  int                  send_buff_len;     /* length of send buffer  */
  char                 recv_buff[256];    /* receive buffer         */
  int                  recv_buff_len;     /* length of recv buffer  */
  int                  cmd_data_len;      /* length of command data */
  int                  i;                 /* counter variable       */
```

```
 EC_struct             EC;               /* error code structure   */
 cmd_struct            Cmd;              /* struct for QTACTLDV    */

 memcpy(device,argv[1],10);              /* copy device name       */

/*********************************************************************/
/*  OPEN connection                                                  */
/*********************************************************************/
 send_buff_len = 0;                      /* no send buffer         */
 recv_buff_len = 0;                      /* no receive buffer      */
 cmd_data_len = 0;                       /* no command data        */
 EC.Edata.Bytes_Provided = 32;           /* No exceptions          */

 QTACTLDV(device,                       /* device name             */
          FUNOPEN,                      /* requested function      */
          send_buff,                    /* send buffer             */
          send_buff_len,                /* length of send buffer   */
          recv_buff,                    /* receive buffer          */
          recv_buff_len,                /* length of receive buffer */
          CTLD0100,                     /* command format          */
          &Cmd,                         /* command data            */
          cmd_data_len,                 /* length of command data  */
          &EC);                         /* Error Code              */

if (EC.Edata.Bytes_Available>0)   /* If there was an error         */
{
  /* Handle the error                                              */
}

/*********************************************************************/
/*  Send Diagnostic command                                          */
/*********************************************************************/
 send_buff_len = 0;                      /* no send buffer         */
 recv_buff_len = 0;                      /* no recv buffer         */
 cmd_data_len = sizeof(Cmd);             /* size of command struct */
 EC.Edata.Bytes_Provided = 32;           /* No exceptions          */
 Cmd.data.Data_transfer_direction = XFRNONE;  /* No data transfer  */
 Cmd.data.Requested_transfer_length = 0;      /* 0 transfer length */
 Cmd.data.Ignore_length_errors = RPTLERR;     /* report length errs */
 Cmd.data.Command_timeout = 600;              /* 10 minute timeout */
 Cmd.data.Type_of_command = CMDSCSI;          /* SCSI command      */
 Cmd.data.Offset_to_command_string = 32;      /* offset 32         */
 Cmd.data.Length_of_command_string = 6;       /* 6 byte command    */
 Cmd.data.Reserved1=0;                        /* reserved          */
 memcpy(&Cmd.cmd_str, SNDDIAG, 6);            /* command string    */

 QTACTLDV(device,                /* device name                    */
          FUNCMD,                /* requested function             */
          send_buff,             /* send buffer                    */
          send_buff_len,         /* length of send buffer          */
          recv_buff,             /* receive buffer                 */
          recv_buff_len,         /* length of receive buffer       */
          CTLD0100,              /* command format                 */
          &Cmd,                  /* command data                   */
          cmd_data_len,          /* length of command data         */
          &EC);                  /* Error code                     */

if (EC.Edata.Bytes_Available>0)   /* If there was an error         */
{
  /* See what message was returned                                 */
  if (strncmp(EC.Edata.Exception_Id,"CPF67C8",7)==0) /* Command
                                                        failed msg  */
  {
    /*****************************************************************/
    /* Check the data returned with CPF67C8                          */
    /*****************************************************************/
```

```
if (strncmp(EC.reason_cd,"\x02\xC0", 2) == 0) /* Device detected
                                                        error   */
{
  /* Check the SCSI completion status                           */
  if (EC.reason_cd[2]=='\x02')      /* Check condition status   */
  {
    /*************************************************************/
    /*  Send Request Sense command                               */
    /*************************************************************/
    send_buff_len = 0;               /* no send buffer          */
    recv_buff_len = 18;              /* length of recv buffer   */
    cmd_data_len = sizeof(Cmd);      /* size of command struct  */
    Cmd.data.Data_transfer_direction = XFRRECV; /* receive      */
    Cmd.data.Requested_transfer_length = 18;    /* 18 bytes     */
    Cmd.data.Ignore_length_errors = IGNLERR; /* ignore length
                                                       errors   */
    Cmd.data.Command_timeout = 60;          /* 60 sec timeout   */
    Cmd.data.Type_of_command = CMDSCSI;     /* SCSI command     */
    Cmd.data.Offset_to_command_string = 32; /* offset 32        */
    Cmd.data.Length_of_command_string = 6;  /* 6 byte cmd       */
    Cmd.data.Reserved1=0;                 /* reserved           */
    memcpy(&Cmd.cmd_str, SNDRSNS, 6);       /* command string   */

    EC.Edata.Bytes_Provided = 32;    /* No exceptions           */

    QTACTLDV(device,         /* device name                     */
             FUNCMD,         /* requested function              */
             send_buff,      /* send buffer                     */
             send_buff_len,  /* length of send buffer           */
             recv_buff,      /* receive buffer                  */
             recv_buff_len,  /* length of receive buffer        */
             CTLD0100,       /* command format                  */
             &Cmd,           /* command data                    */
             cmd_data_len,   /* length of command data          */
             &EC);           /* Error code                      */

    if (EC.Edata.Bytes_Available>0)   /* If there was an error  */
    {
      /* Handle error on request sense command                  */
    }
    else
    {
      /* Parse the request sense data to determine what action  */
      /* to take.                                               */
    }
  }
  else if (EC.reason_cd[2]=='\x08')  /* Busy status             */
  {
    /* Try the command again later                              */
  }
  else                       /* Unexpected completion status    */
  {
    /* Send error message for unexpected completion status      */
  }
}

else if (strncmp(EC.reason_cd,"\x02\xC1\x00", 3) == 0)
                                     /* Selection timeout     */
{
  /* Send message that device might be powered off              */
}

/* Add else if for the other reason codes here                 */

else
{
  /* Send error message for unexpected reason code             */
```

```
    }
  }

  else
  {
    /* Handle other messages                                    */
  }
}
else
{
  /* No error                                                   */
}

/*****************************************************************/
/*  CLOSE connection                                             */
/*****************************************************************/
 send_buff_len = 0;                       /* no send buffer      */
 recv_buff_len = 0;                       /* no receive buffer   */
 cmd_data_len = 0;                        /* no command data     */
 EC.Edata.Bytes_Provided = 32;            /* No exceptions       */

 QTACTLDV(device,                        /* device name          */
          FUNCLOS,                       /* requested function   */
          send_buff,                     /* send buffer          */
          send_buff_len,                 /* length of send buffer*/
          recv_buff,                     /* receive buffer       */
          recv_buff_len,                 /* length of receive buffer */
          CTLD0100,                      /* command format       */
          &Cmd,                          /* command data         */
          cmd_data_len,                  /* length of command data */
          &EC.Edata);                    /* Error code           */

if (EC.Edata.Bytes_Available>0)   /* If there was an error       */
{
  /* Handle the error                                           */
}

}
```

Example 2: Displaying the firmware level of a tape device

```
/* Usage example 2 for QTACTLDV API                              */
/*                                                               */
/*                                                               */

#include <qtactldv.h>                          // Control Device API
#include <qusec.h>                             // Error code header
#include <string.h>                            // String Header File
#include <stdio.h>                             // Standard I/O Header
#include <stdlib.h>                            // Standard Library Header
#include <except.h>                            // Exception & cancel declares

/*****************************************************************************/
/* Type definitions                                                          */
/*****************************************************************************/
// Define the command structure for sending commands using the QTACTLDV API.
typedef struct {                               // Command Structure
  Qta_CTLD0100_t      hdr;                     // Header
  char                cmd_str[6];              // Command String
} ctldv_cmd_t;

/*****************************************************************************/
/* Entry point to program.                                                   */
/*****************************************************************************/
int main (int argc, char *argv[])
{
  char                device[10];              // Device name to get FM level
```

```
int                 deviceLen;                // Length of device name to cop
char                send_buff[1];             // Send buffer
int                 send_buff_len;            // Length of send buffer
char                recv_buff[50];            // Receive buffer
int                 recv_buff_len;            // Length of receive buffer
ctldv_cmd_t         ctldv_cmd;                // Command variable
int                 ctldv_cmd_len;            // Length of command string
char                tempChar;                 // Used to conver ASCII to EBCD
Qus_EC_t            EC;                       // Error code for qtactldv
char                code[4];                  // EBCDIC code level
int                 i;                        // Iterator to move ASCII to EB

EC.Bytes_Provided = 0;                        // Return errors to user
memset(device, ' ', sizeof(device));          // Set to blanks
deviceLen=strlen(argv[1]);
if (deviceLen>10)
  deviceLen=10;
memcpy(device, argv[1], deviceLen);           // Copy up to 10 chars

memset(code, ' ', sizeof(code));              // Clear code level

/*****************************************************************************/
/* Open the pipe.                                                            */
/*****************************************************************************/
send_buff_len = 0;                            // No send buffer
recv_buff_len = 0;                            // No receive buffer
ctldv_cmd_len = 0;                            // No command

#pragma exception_handler(PipeFailed, 0, 0, _C2_MH_ESCAPE, _CTLA_HANDLE)
QTACTLDV(device,                              // Device name
         FUNOPEN,                             // Function requested
         send_buff,                           // Send buffer
         send_buff_len,                       // Send buffer length
         recv_buff,                           // Receive buffer
         recv_buff_len,                       // Receive buffer length
         CTLD0100,                            // Command structure
         &ctldv_cmd,                          // Command data
         ctldv_cmd_len,                       // Command data length
         &EC);                                // Error structure
#pragma disable_handler

/*****************************************************************************/
/*                          Get the drive VPD                                */
/*****************************************************************************/
recv_buff_len = 16;                           // Receive buffer for VPD
send_buff_len = 0;                            // No send buffer
ctldv_cmd_len = 32 + 6;                       // Reserve command size
ctldv_cmd.hdr.Command_timeout = 600;          // 10 minute timeout
ctldv_cmd.hdr.Type_of_command = 0;            // SCSI Command
ctldv_cmd.hdr.Offset_to_command_string = sizeof(ctldv_cmd.hdr); // Offset 32
ctldv_cmd.hdr.Length_of_command_string = 6;   // 6 byte command
ctldv_cmd.hdr.Reserved1 = 0;                  // Reserved
ctldv_cmd.hdr.Data_transfer_direction=XFRRECV; // Receiving inquiry data
ctldv_cmd.hdr.Requested_transfer_length=16;   // Number of bytes to transfer
ctldv_cmd.hdr.Ignore_length_errors = RPTLERR; // Report length errors
memset(ctldv_cmd.cmd_str, 0x00, 6);           // clear command
ctldv_cmd.cmd_str[0] = 0x12;                  // set to Inquiry command
ctldv_cmd.cmd_str[1] = 0x01;                  // set EVPD mode
ctldv_cmd.cmd_str[2] = 0x03;                  // Set code page - VPD
ctldv_cmd.cmd_str[4] = 0x10;                  // Allocation length

#pragma exception_handler(PipeClose, 0, 0, _C2_MH_ESCAPE, _CTLA_HANDLE)
QTACTLDV(device,                              // Device name
         FUNCMD,                              // Function requested
         send_buff,                           // Send buffer
         send_buff_len,                       // Send buffer length
         recv_buff,                           // Receive buffer
```

```
          recv_buff_len,                        // Receive buffer length
          CTLD0100,                             // Command structure
          &ctldv_cmd,                           // Command data
          ctldv_cmd_len,                        // Command data length
          &EC);                                 // Error structure
  #pragma disable_handler

  /*****************************************************************************/
  /*                       Convert the level to EBCDIC                         */
  /*****************************************************************************/
  for (i = 0; (i < sizeof(code)); i++)
  {
     tempChar = recv_buff[12+i];                // Code offest in VPD data
     if (tempChar < 0x41)                       // is it a number?
       tempChar = tempChar - 0x30 + 0xF0;       // ASCII to EBCDII 0-9
     else {
       tempChar = tempChar & 0xDF;              // Convert to ASCII uppercase
       if (tempChar < 0x4A)                     // is char < J ?
         tempChar = tempChar - 0x41 + 0xC1;     // ASCII to EBCDII A-I
       else if (tempChar < 0x53)                // is char < S
         tempChar = tempChar - 0x4A + 0xD1;     // ASCII to EBCDII J-R
       else
         tempChar = tempChar - 0x53 + 0xE2;     // ASCII to EBCDII S-Z
     }
     code[i] = tempChar;                        // Output the EBCDIC char
  }
  printf("The code level is: %s\n", code);

  /*****************************************************************************/
  /*  Close the pipe.                                                          */
  /*****************************************************************************/
  PipeClose:
  send_buff_len = 0;                            // No send buffer
  recv_buff_len = 0;                            // No receive buffer
  ctldv_cmd_len = 0;                            // No command

  #pragma exception_handler(PipeFailed, 0, 0, _C2_MH_ESCAPE, _CTLA_HANDLE)
  QTACTLDV(device,                              // Device name
           FUNCLOS,                             // Function requested
           send_buff,                           // Send buffer
           send_buff_len,                       // Send buffer length
           recv_buff,                           // Receive buffer
           recv_buff_len,                       // Receive buffer length
           CTLD0100,                            // Command structure
           &ctldv_cmd,                          // Command data
           ctldv_cmd_len,                       // Command data length
           &EC);                                // Error structure
  #pragma disable_handler

  PipeFailed:
  return 0;                                     // return to user
}
```

Examples: Processing data queue entries

Your program can use different methods to process data queue entries.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Example 1: Waiting for up to 2 hours to process data queue entries

In this example, Program B specifies to wait up to 2 hours (7200 seconds) to receive an entry from the data queue. Program A sends an entry to data queue DTAQ1 in library QGPL. If program A sends an entry within 2 hours, program B receives the entries from this data queue. Processing begins immediately.

If 2 hours elapse without program A sending an entry, program B processes the time-out condition because the field length returned is 0. Program B continues receiving entries until this time-out condition occurs. The programs are written in CL; however, either program could be written in any high-level language.

The data queue is created with the following command:

```
CRTDTAQ DTAQ(QGPL/DTAQ1) MAXLEN(80)
```

In this example, all data queue entries are 80 bytes long.

In program A, the following statements relate to the data queue:

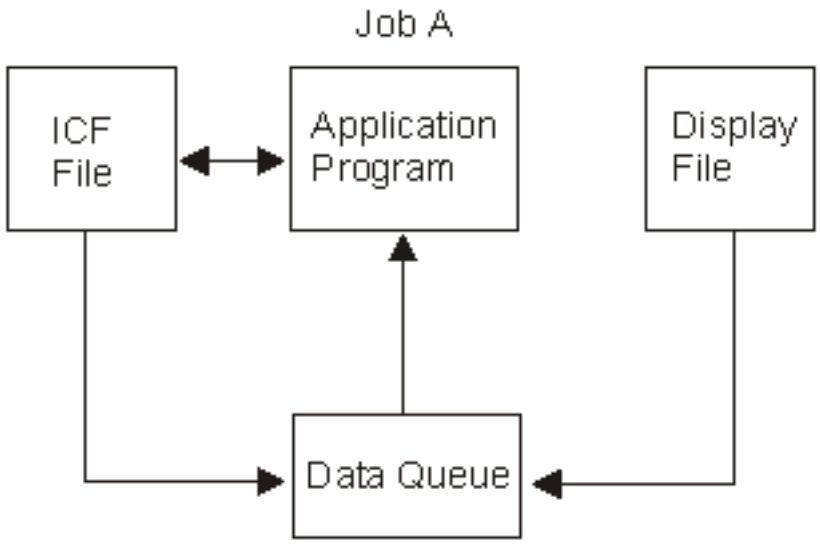
```
PGM
DCL  &FLDLEN  *DEC  LEN(5 0)  VALUE(80)
DCL  &FIELD  *CHAR  LEN(80)
.
.(determine data to be sent to the queue)
.
CALL  QSNDDTAQ  PARM(DTAQ1 QGPL &FLDLEN &FIELD)
.
.
.
```

In program B, the following statements relate to the data queue:

```
PGM
 DCL  &FLDLEN  *DEC  LEN(5 0)  VALUE(80)
 DCL  &FIELD  *CHAR  LEN(80)
 DCL  &WAIT  *DEC  LEN(5 0)  VALUE(7200)   /* 2 hours */
 .
 .
 .
LOOP:   CALL QRCVDTAQ  PARM(DTAQ1
          QGPL &FLDLEN &FIELD &WAIT)
IF (&FLDLEN *NE 0) DO   /* Entry received */
 .
 . (process data from data queue)
 .
   GOTO LOOP /* Get next entry from data queue */
   ENDDO
 .
 . (no entries received for
          2 hours; process time-out condition)
 .
```

Example 2: Processing data queue entries from a display file and an ICF file

The following example is different from the usual use of data queues because there is only one job. The data queue serves as a communications object within the job rather than between two jobs.



In this example, a program is waiting for input from a display file and an ICF file. Instead of alternately waiting for one and then the other, a data queue is used to allow the program to wait on one object (the data queue). The program calls QRCVDTAQ and waits for an entry to be placed on the data queue that was specified on the display file and the ICF file. Both files specify the same data queue. Two types of entries are put on the queue by display data management and ICF data management support when the data is available from either file. ICF file entries start with `*ICFF` and display file entries start with `*DSPF`.

The display file or ICF file entry that is put on the data queue is 80 characters in length and contains the field attributes described in the following table. Therefore, the data queue that is specified using the CRTDSPF, CHGDSPF, OVRDSPF, CRTICFF, CHGICFF, and OVRICFF commands must have a length of at least 80 characters.

Position (and Data Type)	**Description**
1 through 10 (character)	The type of file that placed the entry on the data queue. This field will have one of two values: `*ICFF` for ICF file `*DSPF` for display file If the job receiving the data from the data queue has only one display file or one ICF file open, then this is the only field needed to determine what type of entry has been received from the data queue.
11 through 12 (binary)	The unique identifier for the file. The value of the identifier is the same as the value in the open feedback area for the file. This field should be used by the program receiving the entry from the data queue only if there is more than one file with the same name placing entries on the data queue.
13 through 22 (character)	The name of the display file or ICF file. This is the name of the file actually opened, after all overrides have been processed, and is the same as the file name found in the open feedback area for the file. This field should be used by the program receiving the entry from the data queue only if there is more than one display file or ICF file that is placing entries on the data queue.

Position (and Data Type)	Description
23 through 32 (character)	The library where the file is located. This is the name of the library, after all overrides have been processed, and is the same as the library name found in the open feedback area for the file. This field should be used by the program receiving the entry from the data queue only if there is more than one display file or ICF file that is placing entries on the data queue.
33 through 42 (character)	The program device name, after all overrides have been processed. This name is the same as that found in the program device definition list of the open feedback area. For file type *DSPF, this is the name of the display device where the command or Enter key was pressed. For file type *ICFF, this is the name of the program device where data is available. This field should be used by the program receiving the entry from the data queue only if the file that placed the entry on the data queue has more than one device or session invited prior to receiving the data queue entry.
43 through 80 (character)	Reserved.

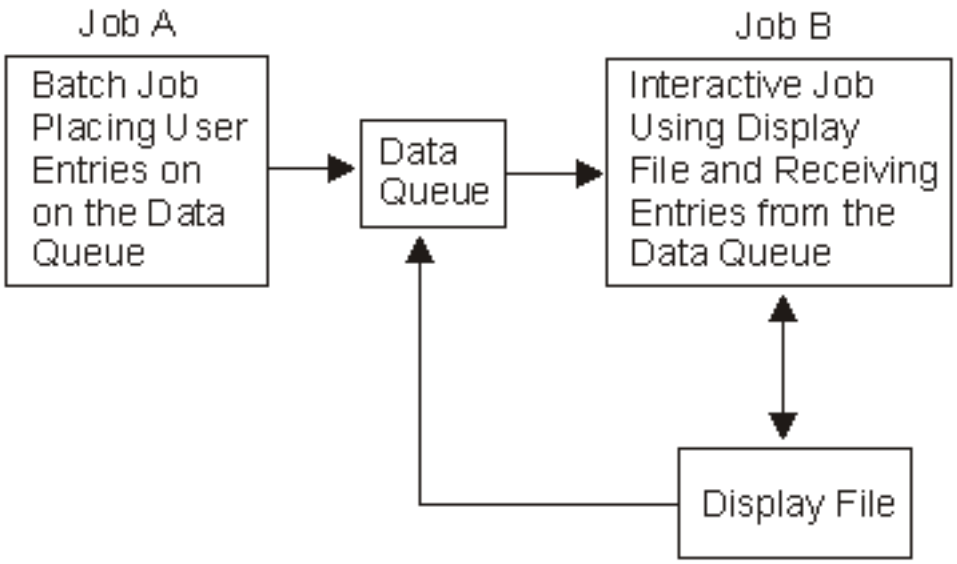
The following example shows coding logic that the application program previously described might use:

```
  .
  .
  .
  .
  OPEN DSPFILE ...
       /* Open the Display file. DTAQ parameter specified on*/
       /* CRTDSPF, CHGDSPF, or OVRDSPF for the file.        */

  OPEN ICFFILE ...
       /* Open the ICF file. DTAQ parameter specified on    */
       /* CRTICFF, CHGICFF, or OVRICFF for the file.        */

 .
 .
 DO
   WRITE DSPFILE
        /* Write with Invite for the Display file           */
   WRITE ICFFILE
        /* Write with Invite for the ICF file               */

   CALL QRCVDTAQ
        /* Receive an entry from the data queue specified   */
        /* on the DTAQ parameters for the files. Entries    */
        /* are placed on the data queue when the data is    */
        /* available from any invited device or session     */
        /* on either file.                                  */
        /* After the entry is received, determine which file */
        /* has data available, read the data, process it,   */
        /* invite the file again and return to process the  */
        /* next entry on the data queue.                    */
   IF 'ENTRY TYPE' FIELD = '*DSPF     ' THEN
                                 /*  Entry is from display */
     DO                          /*  file. Since this entry*/
                                 /*  does not contain the  */
                                 /* data received, the data*/
                                 /* must be read from the  */
                                 /* file before it can be  */
       READ DATA FROM DISPLAY FILE /* processed.            */
       PROCESS INPUT DATA FROM DISPLAY FILE
```

```
      WRITE TO DISPLAY FILE       /* Write with Invite      */
    END
  ELSE                            /* Entry is from ICF      */
                                  /*  file. Since this entry*/
                                  /* does not contain the   */
                                  /* data received, the data*/
                                  /* must be read from the  */
                                  /* file before it can be  */
                                  /* processed.             */
      READ DATA FROM ICF  FILE
      PROCESS INPUT DATA FROM ICF FILE
      WRITE TO ICF FILE           /* Write with Invite      */
  LOOP BACK TO RECEIVE ENTRY FROM DATA QUEUE
.
.
.
END
```

Example 3: Processing entries from a data queue and a display file

In the following example, the program in Job B is waiting for input from a display file that it is using and for input to arrive on the data queue from Job A. Instead of alternately waiting for the display file and then the data queue, the program waits for one object, the data queue.



The program calls QRCVDTAQ and waits for an entry to be placed on the data queue that was specified on the display file. Job A is also placing entries on the same data queue. There are two types of entries put on this queue, the display file entry and the user-defined entry. The display file entry is placed on the data queue by display data management when data is available from the display file. The user-defined entry is placing on the data queue by Job A.

The structure of the display file entry is described in the previous example.

The structure of the entry placed on the queue by Job A is defined by the application programmer.

The following example shows coding logic that the application program in Job B might use:

```
 .
 .
 .
 .
 OPEN DSPFILE ...
           /* Open the Display file. DTAQ parameter specified on*/
           /* CRTDSPF, CHGDSPF, or OVRDSPF for the file.        */


.
.
DO
  WRITE DSPFILE   /* Write with Invite for the Display file      */

  CALL QRCVDTAQ
           /* Receive an entry from the data queue specified    */
           /* on the DTAQ parameter for the file. Entries       */
           /* are placed on the data queue either by Job A or   */
           /* by display data management when data is           */
           /* available from any invited device on the display  */
           /* file.                                             */
           /* After the entry is received, determine what type  */
           /* of entry it is, process it, and return to receive */
           /* the next entry on the data queue.                 */
  IF 'ENTRY TYPE' FIELD = '*DSPF     ' THEN
                                      /* Entry is from display */
    DO                                /*  file. Since this entry*/
                                      /* does not contain the   */
                                      /* data received, the data*/
                                      /* must be read from the  */
                                      /* file before it can be  */
      READ DATA FROM DISPLAY FILE     /* processed.             */
      PROCESS INPUT DATA FROM DISPLAY FILE
      WRITE TO DISPLAY FILE
                                       /* Write with Invite     */
    END
  ELSE                                 /* Entry is from Job A.  */
                                       /* This entry contains   */
                                       /* the data from Job A,  */
                                       /* so no read is required*/
                                       /* before processing the */
                                       /* data.                 */
      PROCESS DATA QUEUE ENTRY FROM JOB A
  LOOP BACK TO RECEIVE ENTRY FROM DATA QUEUE
.
.
.
END
```

Example: Using environment variables

This ILE C program displays the value of an environment variable and sets the environment variable to a new value.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program.

Call this program with one parameter to display the environment variable specified by that parameter. Call this program with two parameters to set the environment variable specified by the first parameter to the value specified by the second parameter.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  Display the value of an environment variable and       */
/*            then set the environment variable to a new value.      */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: getenv(), putenv()                                     */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

#define BUFLEN 1024

int main(int argc, char *argv[])
{
    int   num=0;              /* counter                           */
    int   rc;                 /* API return code                   */
    int   l1, l2;             /* lengths of the two parameters     */
    char  *envvar=NULL;       /* pointer to an environment variable*/
    char  **envvaridx=NULL;   /* pointer to an envvar pointer      */
    char  envstring[BUFLEN];
                              /* buffer to construct putenv request*/

                              /* Show a small usage message.       */
    if ((argc != 2) && (argc != 3)) {
        printf("Usage: %s <ENV_VAR> <new_value>\n OR \n"
               "Usage: %s <ENV_VAR>\n", argv[0], argv[0]);
        printf("Sets an environment variable to a user requested\n"
            "value\n"
            "OR\nDisplays the value of a single environment variable\n");
        exit(1);
      }

    if (argc == 2) {
        /* Called just to display the environment variables.       */
        envvar = getenv(argv[1]);
        if (envvar == NULL) {
            printf("No environment variable %s set\n",
                   argv[1]);
          }
        else {
            printf("Environment variable: %s\n", envvar);
          }
        return 0;
      }

    /* ELSE, called to set an environment variable.                */

    /* Check the size of the parameters and construct a string of  */
    /* the form "VAR=string" which is valid input to putenv.       */
    l1 = strlen(argv[1]);
    l2 = strlen(argv[2]);
    if (l1+l2+2 >= BUFLEN) {
        printf("Only 1024 characters total allowed for this test\n");
        exit(1);
      }
    memcpy(envstring, argv[1], l1);
    envstring[l1] = '=';
    memcpy(&envstring[l1+1], argv[2], l2);
    envstring[l1+l2+1]='\0';
```

```
    /* Now that the string is built, let's see if the environment  */
    /* variable was already set.                                   */
    envvar = getenv(argv[1]);
    if (envvar == NULL) {
        printf("Setting NEW environment variable %s\n",
               envstring);
      }
    else {
        printf("Resetting OLD environment variable from: %s\n to %s\n",
               envvar, envstring);
      }

    /* Now actually set the environment variable.                  */
    rc = putenv(envstring);
    if (rc < 0) {
        printf("putenv failed, errno = %d\n", errno);
        return -1;
      }
    printf("Environment variable set\n");
    return 0;
}
```

Examples: Using ILE Common Execution Environment APIs

These ILE COBOL and ILE RPG programs call the ILE Common Execution Environment (CEE) APIs for date conversions.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Example in ILE COBOL

```
       PROCESS NOMONOPRC.
      ************************************************************
      *
      * This sample ILE COBOL program demonstrates how to call the
      * Common Execution Environment (CEE) Date APIs.  The program
      * accepts two parameters.  The first is the date in character
      * form and the second the format of the date.  For instance
      * CALL CEEDATES ('10131955' 'MMDDYYYY') causes the program
      * to treat the date as October 13 1955).
      *
      * The program then displays on the console the numeric day of
      * the week for that date (Sunday = 1) and the named day of
      * week for that date.
      *
      ************************************************************
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CEEDATES.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
         SPECIAL-NAMES.
         LINKAGE TYPE PROCEDURE FOR "CEEDAYS" USING ALL DESCRIBED,
         LINKAGE TYPE PROCEDURE FOR "CEEDYWK" USING ALL DESCRIBED,
         LINKAGE TYPE PROCEDURE FOR "CEEDATE" USING ALL DESCRIBED.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  Lilian-Date         PIC S9(9)  BINARY.
       01  Day-of-Week-Numeric PIC S9(9)  BINARY.
       01  Day-of-Week-Alpha   PIC X(10).
       01  Day-of-Week-Format  PIC X(10)         VALUE "Wwwwwwwwwz".
       LINKAGE SECTION.
       01  Sample-Date         PIC X(8).
```

```
 01  Date-Format         PIC X(8).
 PROCEDURE DIVISION USING Sample-Date, Date-Format.
 SAMPLE.
*
* Convert formatted date to Lilian date
*
     CALL "CEEDAYS" USING Sample-Date
                          Date-Format
                          Lilian-Date
                          OMITTED.
*
* Get numeric day of week from Lilian date
*
     CALL "CEEDYWK" USING Lilian-Date
                          Day-of-Week-Numeric
                          OMITTED.
*
* Get day of week from Lilian date
*
     CALL "CEEDATE" USING Lilian-Date
                          Day-of-Week-Format
                          Day-of-Week-Alpha
                          OMITTED.
     DISPLAY "Day of week = " Day-of-Week-Numeric UPON CONSOLE.
     DISPLAY "Day of week = " Day-of-Week-Alpha UPON CONSOLE.
     STOP RUN.
```

Example in ILE RPG

```
D************************************************************
D*
D* This sample ILE RPG program demonstrates how to call the
D* Common Execution Environment (CEE) Date APIs.  The program
D* accepts two parameters.  The first is the date in character
D* form and the second the format of the date.  For instance
D* CALL CEEDATES ('10131955' 'MMDDYYYY') causes the program
D* to treat the date as October 13 1955).
D*
D* The program must be compiled with DFTACTGRP(*NO)
D*
D* The program then displays on the console the numeric day of
D* the week for that date (Sunday = 1) and the named day of
D* week for that date.
D*
D************************************************************
DLilianDate       s             10i 0
DDayOfWkN         s             10i 0
DDayOfWkA         s             10
DDayOfWkFmt       s             10    inz('Wwwwwwwwwz')
C     *entry        plist
C                   parm                    SampleDate        8
C                   parm                    DateFormat        8
C*
C* Convert formatted date to Lilian date
C*
C                   callb(d)  'CEEDAYS'
C                   parm                    SampleDate
C                   parm                    DateFormat
C                   parm                    LilianDate
C                   parm                    *OMIT
C*
C* Get numeric day of week from Lilian date
C*
C                   callb(d)  'CEEDYWK'
C                   parm                    LilianDate
C                   parm                    DayOfWkN
C                   parm                    *OMIT
```

```
C*
C* Get day of week from Lilian date
C*
C                   callb(d)  'CEEDATE'
C                   parm                    LilianDate
C                   parm                    DayOfWkFmt
C                   parm                    DayOfWkA
C                   parm                    *OMIT
C*
C     DayOfWkN      dsply
C     DayOfWkA      dsply
C                   eval      *inlr = '1'
C                   return
```

Example: Using generic terminal APIs

These example programs implement a generic terminal and a simple interpreter.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Terminal program

This example program starts and runs a generic terminal.

It demonstrates the use of the generic terminal functions Qp0zStartTerminal(), Qp0zRunTerminal(), Qp0zEndTerminal(), and Qp0zGetTerminalPid().

Use the Create Bound C Program (CRTBNDC) command to create this program (see "Creating the terminal and interpreter programs" on page 402).

Call this program with no parameters (see "Calling the terminal program" on page 402).

```
/* Includes */
#include <qp0ztrml.h>
#include <qp0z1170.h>
#include <stdlib.h>
#include <stdio.h>

/* Constants */
#define NUM_PREDEFINED_ENVS 2

/* Global Variables */
extern char **environ;

int main (int argc, char *argv[])
{
  char *args[2];             /* Argument array */
  int envCount;              /* Count of currently defined env vars */
  int index;                 /* Loop index */
  char **envp;               /* For walking environ array */
  char **envs;               /* Environment variable array */
  Qp0z_Terminal_T handle;    /* Terminal handle */
  Qp0z_Terminal_Attr_T ta;   /* Terminal attributes */
  pid_t pid;                 /* Process ID of interpreter */
  int rc;                    /* Return code */

  /*********************************************************************/
  /* Build the argument array.  */
  /*********************************************************************/

  args[0] = "/QSYS.LIB/QGPL.LIB/ECHOINT.PGM";
  args[1] = NULL;

  /*********************************************************************/
```

```
  /* Build the environment variable array. */
  /*********************************************************************/

  /* Make sure environ is set in this activation group. */
  Qp0zInitEnv();

  /* Count the number of environment variables currently defined in this
     process.  Qp0zStartTerminal() will make sure the interpreter
     process does not have too many environment variables. */
  for (envCount = 0, envp = environ; *envp != NULL; ++envp, ++envCount);

  /* Allocate storage for the environment variable array. */
  envs = (char **)malloc(sizeof(char *) *
                         (envCount + NUM_PREDEFINED_ENVS));
  if (envs == NULL) {
    perror("malloc() failed");
    exit(1);
  }

  /* Copy the current environment variables to the array. */
  for (index = 0; environ[index] != NULL; ++index) {
    envs[index] = environ[index];
  }

  /* Set QIBM_USE_DESCRIPTOR_STDIO variable for using descriptors.  This
     will override the current value of the variable. */
  envs[index++] = "QIBM_USE_DESCRIPTOR_STDIO=Y";

  /* Null terminate array of environment variables. */
  envs[index] = NULL;

  /*********************************************************************/
  /* Set the terminal attributes.  */
  /*********************************************************************/

  memset(&ta, '\0', sizeof(Qp0z_Terminal_Attr_T));
  ta.Buffer_Size = 8196;
  ta.Inherit.pgroup = SPAWN_NEWPGROUP;
  ta.Title = "Echo Interpreter";
  ta.Cmd_Key_Line1 = "F3=Exit F9=Retrieve";
  ta.Cmd_Key_Line2 = "F17=Top F18=Bottom";

  /*********************************************************************/
  /* Start and run the terminal.  */
  /*********************************************************************/

  /* Start the terminal. */
  if (Qp0zStartTerminal(
handle, args, envs, ta) != 0) {
    perror("Qp0zStartTerminal() failed");
    exit(1);
  }

  /* Get the PID of the interpreter process. */
  if (Qp0zGetTerminalPid(handle, &pid) != 0) {
    perror("Qp0zGetTerminalPid() failed");
    exit(1);
  }

  /* Run the terminal. */
  rc = Qp0zRunTerminal(handle);
  switch (rc) {
    case QP0Z_TERMINAL_F12:
    case QP0Z_TERMINAL_F3:
    case QP0Z_TERMINAL_ENDED:
      /* Do nothing */
      break;
```

```
    default:
      perror("Qp0zRunTerminal() failed");
      exit(1);
      break;
  }

  /* End the terminal. */
  Qp0zEndTerminal(handle);

  return 0;
}
```

Interpreter program

This example program is a simple echo interpreter that is used by the terminal program (see "Terminal program" on page 399).

Use the Create Bound C Program (CRTBNDC) command to create this program (see "Creating the terminal and interpreter programs" on page 402).

```
/* Echo interpreter */
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

static void SignalHandler(int);

int main (int argc, char *argv[])
{
  char buffer[8192];         /* Buffer for reading input */
  struct sigaction sigact;   /* Signal action */

  /* Set up a signal handler for SIGHUP.  The terminal
     sends this signal when the user presses F3 to exit. */
  sigemptyset(&sigact.sa_mask);
  sigact.sa_flags = 0;
  sigact.sa_handler = SignalHandler;
  if (sigaction(SIGHUP, &sigact, NULL) != 0) {
    perror("sigaction(SIGHUP) failed.");
    exit(2);
  }

  /* Set up a signal handler for SIGINT.  The terminal sends
     this signal when the user presses SysReq 2. */
  sigemptyset(&sigact.sa_mask);
  sigact.sa_flags = 0;
  sigact.sa_handler = SignalHandler;
  if (sigaction(SIGINT, &sigact, NULL) != 0) {
    perror("sigaction(SIGINT) failed.");
    exit(2);
  }

  /* Switch stdout to use line-mode buffering. */
  setvbuf(stdout, NULL, _IOLBF, 128);
  printf("Echo interpreter starting ...\n");
  printf("Enter text:\n");

  /* Do forever. */
  while (1) {
    /* Read a line from stdin. */
    gets(buffer);

    /* End and clean up any allocated
       resources when stdin is closed. */
    if (feof(stdin)) {
```

```
      printf("Echo interpreter ending ...\n");
      exit(0);
    }

    /* Echo the line to stdout. */
    printf("%s\n", buffer);
  } /* End of while */

  return 0;
}

void
SignalHandler(int signo)
{
  printf("Ending for signal %d\n", signo);
  exit(1);
}
```

Creating the terminal and interpreter programs

The following examples show how to create the example programs ("Terminal program" on page 399 and "Interpreter program" on page 401). These examples assume that the source for the terminal program is member TERMINAL in the file QGPL/QCSRC and that the source for the interpreter program is member INTERPRET in the file QGPL/QCSRC.

Creating the terminal program:

```
CRTBNDC PGM(QGPL/TERMINAL)
        SRCFILE(QGPL/QCSRC)
        SRCMBR(TERMINAL)
        SYSIFCOPT(*IFSIO)
        TEXT('Example Terminal program')
```

Creating the interpreter program:

```
CRTBNDC PGM(QGPL/INTERPRET)
        SRCFILE(QGPL/QCSRC)
        SRCMBR(INTERPRET)
        SYSIFCOPT(*IFSIO)
        TEXT('Example Interpreter program')
```

Calling the terminal program

The following example shows how to start the example program:

```
CALL PGM(QGPL/TERMINAL)
```

Example: Using profile handles

This example shows how to use APIs to generate, change, and release profile handles in a CL program.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/* FUNCTION:  Illustrates how to generate, change, and release   */
/*            profile handles in a CL program.                   */
/*                                                               */
/* LANGUAGE:  CL                                                 */
/*                                                               */
/* APIs USED: QSYGETPH    - Get Profile Handle                   */
/*            QWTSETP     - Set Profile                          */
/*            QSYRLSPH    - Release Profile Handle               */
/*                                                               */
```

```
/*****************************************************************/
/*****************************************************************/

PGM (&USERID &PWD &PWDLEN)

/*-------------------------------------------------------------*/
/* Parameters:                                                 */
/*-------------------------------------------------------------*/

/* 10 Character user ID                                        */
DCL        VAR(&USERID) TYPE(*CHAR) LEN(10)

/* Password  (up to 50 bytes)                                  */
/* This password is case sensitive                             */
DCL        VAR(&PWD) TYPE(*CHAR) LEN(50)

/* Length of the password in binary(4) form (example-- a       */
/* 5 byte password length would be X'00000005)                 */
DCL        VAR(&PWDLEN) TYPE(*CHAR) LEN(4)


/*-------------------------------------------------------------*/
/* Variables needed by this program:                           */
/*-------------------------------------------------------------*/

/* Password CCSID value of -1. The current password level      */
/* for the system is used to determine the CCSID of the        */
/* password.                                                   */
DCL        VAR(&PWDCCSID) TYPE(*CHAR) LEN(4) +
                            VALUE( X'FFFFFFFF')

/* Exceptions will be signalled                                */
DCL        VAR(&ERRCODE)  TYPE(*CHAR) LEN(8) +
                            VALUE( X'0000000000000000')

/* Password for *CURRENT user ID. When *CURRENT is             */
/* specified for the user ID, the password field will be       */
/* ignored.                                                    */
DCL        VAR(&CURPWD)   TYPE(*CHAR) LEN(10) +
                            VALUE('          ')

/* Profile handles returned                                    */
DCL        VAR(&PRFHNDL1) TYPE(*CHAR) LEN(12)
DCL        VAR(&PRFHNDL2) TYPE(*CHAR) LEN(12)

/*-------------------------------------------------------------*/
/* Generate profile handles for the user ID this program       */
/* is currently running under and for the user ID passed       */
/* to this program:                                            */
/*-------------------------------------------------------------*/
CALL       PGM(QSYGETPH) PARM('*CURRENT  ' +
                              &CURPWD  /* Password ignored  +
                                          when *CURRENT is  +
                                          specified       */+
                              &PRFHNDL1)

CALL       PGM(QSYGETPH) PARM(&USERID +
                              &PWD    +
                              &PRFHNDL2 +
                              &ERRCODE   /* Exceptions will   +
                                            be signalled   */ +
                              &PWDLEN    /* Length of pwd  */ +
                              &PWDCCSID) /* Password CCSID */

/*-------------------------------------------------------------*/
/* 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 user ID  */
/* it was originally running under                            */
/*-------------------------------------------------------------*/
CALL       PGM(QWTSETP) PARM(&PRFHNDL1)

/*-------------------------------------------------------------*/
/* The profile handles generated in this program can now    */
/* be released:                                                */
/*-------------------------------------------------------------*/
CALL       PGM(QSYRLSPH) PARM(&PRFHNDL1)
CALL       PGM(QSYRLSPH) PARM(&PRFHNDL2)

ENDPGM
```

Example: Using registration facility APIs

These ILE C programs show how to register an exit point, how to add an exit program to the exit point, and how to call the exit program based on the exit point information that is retrieved.

This example does not include any of the programs that are being called, nor does it show anything but an excerpt of the calling program.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

The first thing to do, after deciding in what program object the exit point is to be placed, is to register that exit point. It is also important to remember that the exit point format defines what the exit program data looks like. This ILE C program registers an exit point named QIBM_QXIC_TSTXPOINTA.

```
/*********************************************************************/
/*  PROGRAM:      RegisterPoint                                      */
/*                                                                   */
/*  LANGUAGE:     ILE C                                              */
/*                                                                   */
/*  DESCRIPTION:  This program registers an exit point in an         */
/*                application.                                       */
/*                                                                   */
/* APIs USED:     QusRegisterExitPoint                               */
/*                                                                   */
/*********************************************************************/
#include <string.h>
#include <qusec.h>
#include <qusrgfa1.h>

/*********************************************************************/
/*  Structure for the control block                                  */
/*********************************************************************/
typedef _Packed struct Control_x{
        int                 Num_Vlen_Recs;
        Qus_Vlen_Rec_4_t    Vlen_Rec_1;
        char                Description[50];
} Control_Rec;


int main () {
```

```
   Qus_EC_t     Error_Code     = {0};

   char         EPnt_Name[20]  = "QIBM_QXIC_TSTXPOINTA";
   char         EPnt_F_Name[8] = "USUSOOOO";
   int          EProg_Number   = -1;
   Control_Rec  EPnt_Controls  = {0};

/***************************************
*** INITIALIZING ALL STRUCTURES::      ***
***************************************/

   Error_Code.Bytes_Provided = sizeof(Error_Code);

   EPnt_Controls.Num_Vlen_Recs = 1;
   EPnt_Controls.Vlen_Rec_1.Length_Vlen_Record = 62;
   EPnt_Controls.Vlen_Rec_1.Control_Key = 8;
   EPnt_Controls.Vlen_Rec_1.Length_Data = 50;
   memcpy( EPnt_Controls.Description , "Example Exit Point" , 17 );

   QusRegisterExitPoint (EPnt_Name,
                         EPnt_F_Name,
                         &EPnt_Controls,
                         &Error_Code);
   if ( Error_Code.Bytes_Available ) {
      printf("\nEXCEPTION : %s",Error_Code.Exception_Id);
      exit (1);
   }
   return(0);
}
```

After an exit point has been registered, exit programs must be added to that point, indicating the possible calls based on runtime conditions. The following ILE C program adds an exit program named TSTXITPROGQGPL to the QIBM_QXIC_TSTXPOINTA exit point that is registered in the previous example.

```
/*********************************************************************/
/*  PROGRAM:      AddProgram                                         */
/*                                                                   */
/*  LANGUAGE:     ILE C                                              */
/*                                                                   */
/*  DESCRIPTION:  This program adds an exit program to a registered */
/*                exit point.                                        */
/*                                                                   */
/* APIs USED:     QusAddExitProgram                                  */
/*                                                                   */
/*********************************************************************/

#include <qusec.h>
#include <qusrgfa1.h>

/*********************************************************************/
/*  Structure for the Exit Program Attributes                        */
/*********************************************************************/

typedef _Packed struct Xit_Att{
        int                  Num_Vlen_Recs;
        Qus_Vlen_Rec_4_t     ADPG_Vlen;
        int                  CCSID;
        char                 Reserved;
} Xit_Attributes;

int main () {

   Qus_EC_t       Error_Code       = {0};

   char           EPnt_Name[20]    = "QIBM_QXIC_TSTXPOINTA";
```

```
   char           EPnt_F_Name[8]   = "USUSOOOO";
   int            EProg_Number     = -1;
   char           Q_EProg_Name[20] = "TSTXITPROGQGPL      ";
   char           EProg_Data[10]   = "EXAMPLE   ";
   int            Len_EProg_Data   = sizeof(EProg_Data);
   Xit_Attributes EProg_Attributes;


   Error_Code.Bytes_Provided=sizeof(Error_Code);

   EProg_Attributes.Num_Vlen_Recs=1;
   EProg_Attributes.ADPG_Vlen.Length_Vlen_Record=16;
   EProg_Attributes.ADPG_Vlen.Control_Key=3;
   EProg_Attributes.ADPG_Vlen.Length_Data=4;
   EProg_Attributes.CCSID = 37;

   QusAddExitProgram (EPnt_Name,
                      EPnt_F_Name,
                      EProg_Number,
                      Q_EProg_Name,
                      EProg_Data,
                      Len_EProg_Data,
                      &EProg_Attributes,
                      &Error_Code);
   if ( Error_Code.Bytes_Available ) {
      printf("\nEXCEPTION : %s",Error_Code.Exception_Id);
      exit (1);
   }
   return(0);
}
```

When you have registered an exit point and have added the exit programs to that exit point, you can do exit program calls from within your application. The information needed to do the calls is obtained from the Retrieve Exit Information API. In the following sample a conditional call is made based on the exit point information.

```
/*********************************************************************/
/*  PROGRAM:      RetrieveAndProcess                                 */
/*                                                                   */
/*  LANGUAGE:     ILE C                                              */
/*                                                                   */
/*  DESCRIPTION:  This is an excerpt of a program that retrieves     */
/*                information on an exit point, and does processing  */
/*                based on that information.                         */
/*                                                                   */
/* APIs USED:     QusRetrieveExitInformation                         */
/*                                                                   */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qusec.h>
#include <qusrgfa2.h>

/*********************************************************************/
/*  Structure for Selection Criteria on the Retrieve                 */
/*********************************************************************/
typedef _Packed struct RTVEI_Select_C_x {
        Qus_Selcrtr_t        Crit;
        Qus_Select_Entry_t   Select_Entry;
        char                 RTV_String[10];
} RTVEI_Select_C;

/*********************************************************************/
/*  Conv_Lib converts the library name to a null terminated string   */
/*********************************************************************/
```

```
char * Conv_Lib(char in_lib[], char *tmp) {
      int x = 0;

      while ( (in_lib[x] != ' ') && x!=10 ) {
         *tmp=in_lib[x++];
         tmp++;
      }
      return(tmp);
}

int main() {

   Qus_EXTI0200_t       *EXTI0200;
   Qus_EXTI0200_Entry_t *EXTI0200_Entry;
   char                 *Pgm_Data;

   Qus_EC_t        Error_Code= {0};

   char            EPnt_Name[20]  = "QIBM_QXIC_TSTXPOINTA";
   char            EPnt_F_Name[8] = "USUSOOOO";
   int             EProg_Number   = -1;

   int             Counter;
   char            *tmp_str;
   char            *lib;

   char            Handle[16]     = "                ";
   int             Length_Of_R_Var;
   char            RTVEI_Format_Name[8];
   RTVEI_Select_C  EProg_Select_C = {0};


/****************************************
*  Initializing Structures              *
****************************************/

   Error_Code.Bytes_Provided = sizeof(Error_Code);

   tmp_str=(char *)malloc(sizeof(char));
   lib=(char *)malloc(sizeof(char));
   EXTI0200=(Qus_EXTI0200_t *) malloc ( ( sizeof( Qus_EXTI0200_t ) +
        sizeof( Qus_EXTI0200_Entry_t ) + MAX_PGM_DATA_SIZE ) * 2 );
   EProg_Select_C.Crit.Number_Sel_Criteria = 1;
   EProg_Select_C.Select_Entry.Size_Entry = 26;
   EProg_Select_C.Select_Entry.Comp_Operator = 1;
   EProg_Select_C.Select_Entry.Start_Pgm_Data = 0;
   EProg_Select_C.Select_Entry.Length_Comp_Data = 10;
   memcpy( EProg_Select_C.RTV_String , "EXAMPLE   " , 10 );

   Length_Of_R_Var = (sizeof( Qus_EXTI0200_t ) +
                sizeof( Qus_EXTI0200_Entry_t ) +
                MAX_PGM_DATA_SIZE) *2;
   memcpy( RTVEI_Format_Name , "EXTI0200" , 8 );

   QusRetrieveExitInformation (Handle,
                               EXTI0200,
                               Length_Of_R_Var,
                               RTVEI_Format_Name,
                               EPnt_Name,
                               EPnt_F_Name,
                               EProg_Number,
                               &EProg_Select_C,
                               &Error_Code);
   if ( Error_Code.Bytes_Available ) {
      printf("\nEXCEPTION : %s",Error_Code.Exception_Id);
      exit (1);
   }
```

```
/*******************************************************
*  Call all of the preprocessing exit programs returned *
*******************************************************/
   Counter=EXTI0200->Number_Programs_Returned;

   while ( Counter-- ) {

     EXTI0200_Entry = (Qus_EXTI0200_Entry_t *) EXTI0200;
     EXTI0200_Entry = (Qus_EXTI0200_Entry_t *)((char *)EXTI0200 +
                       EXTI0200->Offset_Program_Entry);
     Pgm_Data = (char *) EXTI0200_Entry;
     Pgm_Data += EXTI0200_Entry->Offset_Exit_Data;

     Conv_Lib(EXTI0200_Entry->Program_Library,lib);

     sprintf( tmp_str , "CALL  %s/%.10s %.*s" ,
          lib,
          EXTI0200_Entry->Program_Name,
          EXTI0200_Entry->Length_Exit_Data,
          Pgm_Data );
     system( tmp_str );
/*******************************************************
*  This is where Error Handling on the exit program     *
*  would be done.                                       *
*******************************************************/
     if ( Counter ) {
         memcpy(EXTI0200->Continue_Handle,Handle,16);
         QusRetrieveExitInformation(Handle,
                                    EXTI0200,
                                    Length_Of_R_Var,
                                    RTVEI_Format_Name,
                                    EPnt_Name,
                                    EPnt_F_Name,
                                    EProg_Number,
                                    &EProg_Select_C,
                                    &Error_Code);
         if ( Error_Code.Bytes_Available ) {
            printf("\nEXCEPTION : %s",Error_Code.Exception_Id);
            exit (1);
         }
      }
   }

   return(0);
}
```

Example: Using semaphore set and shared memory functions

This example illustrates programs that support the client/server model.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Server program

This program acts as a server to the client program (see "Client program" on page 411). The buffer is a shared memory segment. The process synchronization is done using semaphores.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program.

Call this program with no parameters before calling the client program.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  This program acts as a server to the client program.   */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: semctl(), semget(), semop(),                           */
/*            shmat(), shmctl(), shmdt(), shmget()                   */
/*            ftok()                                                 */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define SEMKEYPATH "/dev/null"  /* Path used on ftok for semget key  */
#define SEMKEYID 1              /* Id used on ftok for semget key    */
#define SHMKEYPATH "/dev/null"  /* Path used on ftok for shmget key  */
#define SHMKEYID 1              /* Id used on ftok for shmget key    */

#define NUMSEMS 2               /* Num of sems in created sem set    */
#define SIZEOFSHMSEG 50         /* Size of the shared mem segment    */

#define NUMMSG 2                /* Server only doing two "receives"
                                   on shm segment                    */

int main(int argc, char *argv[])
{
    int rc, semid, shmid, i;
    key_t semkey, shmkey;
    void *shm_address;
    struct sembuf operations[2];
    struct shmid_ds shmid_struct;
    short  sarray[NUMSEMS];

    /* Generate an IPC key for the semaphore set and the shared      */
    /* memory segment.  Typically, an application specific path and  */
    /* id would be used to generate the IPC key.                     */
    semkey = ftok(SEMKEYPATH,SEMKEYID);
    if ( semkey == (key_t)-1 )
      {
        printf("main: ftok() for sem failed\n");
        return -1;
      }
    shmkey = ftok(SHMKEYPATH,SHMKEYID);
    if ( shmkey == (key_t)-1 )
      {
        printf("main: ftok() for shm failed\n");
        return -1;
      }

    /* Create a semaphore set using the IPC key.  The number of      */
    /* semaphores in the set is two.  If a semaphore set already     */
    /* exists for the key, return an error. The specified permissions*/
    /* give everyone read/write access to the semaphore set.         */

    semid = semget( semkey, NUMSEMS, 0666 | IPC_CREAT | IPC_EXCL );
    if ( semid == -1 )
      {
        printf("main: semget() failed\n");
        return -1;
      }
```

```
/* Initialize the first semaphore in the set to 0 and the        */
/* second semaphore in the set to 0.                             */
/*                                                               */
/* The first semaphore in the sem set means:                     */
/*        '1' --  The shared memory segment is being used.       */
/*        '0' --  The shared memory segment is freed.            */
/* The second semaphore in the sem set means:                    */
/*        '1' --  The shared memory segment has been changed by  */
/*                the client.                                    */
/*        '0' --  The shared memory segment has not been         */
/*                changed by the client.                         */

sarray[0] = 0;
sarray[1] = 0;


/* The '1' on this command is a no-op, because the SETALL command*/
/* is used.                                                      */
rc = semctl( semid, 1, SETALL, sarray);
if(rc == -1)
  {
    printf("main: semctl() initialization failed\n");
    return -1;
  }

/* Create a shared memory segment using the IPC key.  The        */
/* size of the segment is a constant.  The specified permissions */
/* give everyone read/write access to the shared memory segment. */
/* If a shared memory segment already exists for this key,       */
/* return an error.                                              */
shmid = shmget(shmkey, SIZEOFSHMSEG, 0666 | IPC_CREAT | IPC_EXCL);
if (shmid == -1)
  {
    printf("main: shmget() failed\n");
    return -1;
  }

/* Attach the shared memory segment to the server process.       */
shm_address = shmat(shmid, NULL, 0);
if ( shm_address==NULL )
  {
    printf("main: shmat() failed\n");
    return -1;
  }
printf("Ready for client jobs\n");

/* Loop only a specified number of times for this example.       */
for (i=0; i < NUMMSG; i++)
  {
    /* Set the structure passed into the semop() to first wait   */
    /* for the second semval to equal 1, then decrement it to    */
    /* allow the next signal that the client writes to it.       */
    /* Next, set the first semaphore to equal 1, which means     */
    /* that the shared memory segment is busy.                   */
    operations[0].sem_num = 1;
                                /* Operate on the second sem     */
    operations[0].sem_op = -1;
                                /* Decrement the semval by one   */
    operations[0].sem_flg = 0;
                                /* Allow a wait to occur         */

    operations[1].sem_num = 0;
                                /* Operate on the first sem      */
    operations[1].sem_op =  1;
                                /* Increment the semval by 1     */
    operations[1].sem_flg = IPC_NOWAIT;
```

```
                                        /* Do not allow to wait    */

        rc = semop( semid, operations, 2 );
        if (rc == -1)
          {
            printf("main: semop() failed\n");
            return -1;
          }

        /* Print the shared memory contents.                        */
        printf("Server Received : \"%s\"\n", (char *) shm_address);

        /* Signal the first semaphore to free the shared memory.    */
        operations[0].sem_num = 0;
        operations[0].sem_op  = -1;
        operations[0].sem_flg = IPC_NOWAIT;

        rc = semop( semid, operations, 1 );
        if (rc == -1)
          {
            printf("main: semop() failed\n");
            return -1;
        }

      }  /* End of FOR LOOP */

    /* Clean up the environment by removing the semid structure,    */
    /* detaching the shared memory segment, and then performing     */
    /* the delete on the shared memory segment ID.                  */

    rc = semctl( semid, 1, IPC_RMID );
    if (rc==-1)
      {
        printf("main: semctl() remove id failed\n");
        return -1;
      }
    rc = shmdt(shm_address);
    if (rc==-1)
      {
        printf("main: shmdt() failed\n");
        return -1;
      }
    rc = shmctl(shmid, IPC_RMID, &shmid_struct);
    if (rc==-1)
      {
        printf("main: shmctl() failed\n");
        return -1;
      }
return 0;
}
```

Client program

This program acts as a client to the server program (see "Server program" on page 408). The program is run after the message `Ready for client jobs` appears from the server program.

Use the CRTCMOD and CRTPGM commands to create this program.

Call this program with no parameters after calling the server program.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  This program acts as a client to the server program.   */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
```

```
/*                                                                  */
/* APIs USED: semget(), semop(),                                    */
/*            shmget(), shmat(), shmdt()                            */
/*            ftok()                                                */
/*                                                                  */
/********************************************************************/
/********************************************************************/

#include <stdio.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define SEMKEYPATH "/dev/null"  /* Path used on ftok for semget key  */
#define SEMKEYID 1              /* Id used on ftok for semget key    */
#define SHMKEYPATH "/dev/null"  /* Path used on ftok for shmget key  */
#define SHMKEYID 1              /* Id used on ftok for shmget key    */

#define NUMSEMS 2
#define SIZEOFSHMSEG 50

int main(int argc, char *argv[])
{
    struct sembuf operations[2];
    void         *shm_address;
    int semid, shmid, rc;
    key_t semkey, shmkey;

    /* Generate an IPC key for the semaphore set and the shared     */
    /* memory segment.  Typically, an application specific path and */
    /* id would be used to generate the IPC key.                    */
    semkey = ftok(SEMKEYPATH,SEMKEYID);
    if ( semkey == (key_t)-1 )
      {
        printf("main: ftok() for sem failed\n");
        return -1;
      }
    shmkey = ftok(SHMKEYPATH,SHMKEYID);
    if ( shmkey == (key_t)-1 )
      {
        printf("main: ftok() for shm failed\n");
        return -1;
      }

    /* Get the already created semaphore ID associated with key.    */
    /* If the semaphore set does not exist, then it will not be     */
    /* created, and an error will occur.                            */
    semid = semget( semkey, NUMSEMS, 0666);
    if ( semid == -1 )
      {
        printf("main: semget() failed\n");
        return -1;
      }

    /* Get the already created shared memory ID associated with key. */
    /* If the shared memory ID does not exist, then it will not be   */
    /* created, and an error will occur.                             */

    shmid = shmget(shmkey, SIZEOFSHMSEG, 0666);
    if (shmid == -1)
      {
        printf("main: shmget() failed\n");
        return -1;
      }

    /* Attach the shared memory segment to the client process.      */
```

```
shm_address = shmat(shmid, NULL, 0);
if ( shm_address==NULL )
  {
    printf("main: shmat() failed\n");
    return -1;
  }

/* First, check to see if the first semaphore is a zero.  If it  */
/* is not, it is busy right now.  The semop() command will wait  */
/* for the semaphore to reach zero before running the semop().   */
/* When it is zero, increment the first semaphore to show that   */
/* the shared memory segment is busy.                            */
operations[0].sem_num = 0;
                                /* Operate on the first sem      */
operations[0].sem_op =  0;
                                /* Wait for the value to be=0    */
operations[0].sem_flg = 0;
                                /* Allow a wait to occur         */

operations[1].sem_num = 0;
                                /* Operate on the first sem      */
operations[1].sem_op =  1;
                                /* Increment the semval by one   */
operations[1].sem_flg = 0;
                                /* Allow a wait to occur         */

rc = semop( semid, operations, 2 );
if (rc == -1)
  {
    printf("main: semop() failed\n");
    return -1;
  }

strcpy((char *) shm_address, "Hello from Client");

/* Release the shared memory segment by decrementing the in-use  */
/* semaphore (the first one).  Increment the second semaphore to */
/* show that the client is finished with it.                     */
operations[0].sem_num = 0;
                                /* Operate on the first sem      */
operations[0].sem_op =  -1;
                                /* Decrement the semval by one   */
operations[0].sem_flg = 0;
                                /* Allow a wait to occur         */

operations[1].sem_num = 1;
                                /* Operate on the second sem     */
operations[1].sem_op =  1;
                                /* Increment the semval by one   */
operations[1].sem_flg = 0;
                                /* Allow a wait to occur         */

rc = semop( semid, operations, 2 );
if (rc == -1)
  {
    printf("main: semop() failed\n");
    return -1;
  }

/* Detach the shared memory segment from the current process.    */
rc = shmdt(shm_address);
if (rc==-1)
  {
    printf("main: shmdt() failed\n");
    return -1;
```

```
      }

return 0;
}
```

Example: Using SNA/Management Services Transport APIs

This example shows a source and target application in ILE C that use the Systems Network Architecture (SNA) Management Services Transport APIs to send and receive management services data.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Source application program

This source application program sends management services transport requests to a target application program.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:                                                         */
/*  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  */
/*        string "ENDRMTAPP".                                        */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: QNMSTRAP, QNMENDAP, QNMRCVDT,                          */
/*            QNMSNDRQ, QNMCHGMN, QNMENDAP                           */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/* 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 program receives management services transport requests from the source application program. The target application program returns replies if requests specify that replies are needed.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:                                                         */
/* 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. */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: QNMSTRAP, QNMENDAP, QNMREGAP, QNMDRGAP,                */
/*            QNMRCVDT, QNMSNDRP, QNMRCVOC, QRCVDTAQ,                */
/*            QNMENDAP                                               */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/* 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, that is, 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            */
  }
}
```

Example: Using source debugger APIs

The ILE source debugger APIs allow an application developer to write a debugger for ILE programs.

You might ask why this would ever be done when an ILE debugger is provided with the IBM i operating system. There are several reasons why an application developer might want to use these APIs to write a different ILE debugger:

- A debugger running on a workstation can be built to debug ILE programs running on the system. This allows a debugger to take advantage of Windows and other easy-to-use interfaces available on the workstation. The workstation debugger can communicate with the code running on the system. The code running on the system can use the debugger APIs.
- The writer of an ILE compiler might want to write a debugger to take advantage of the ILE languages. The IBM i debugger is a more general-purpose debugger that is made for all ILE languages.
- A debugger can be written with functions not available on the IBM i ILE debugger.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Source debugger APIs overview

The ILE source debugger APIs can be divided into several groups. These include APIs that:

- Start and end the debug session
- Add programs and modules to debug
- Manipulate text views in a program
- Add and remove breakpoints, steps, and so on

Besides APIs, there are two user exits that get called:

- The Source Debug program gets called when the Start Debug (STRDBG), Display Module Source (DSPMODSRC), and End Debug (ENDDBG) CL commands are entered.
- The Program Stop Handler gets called when an ILE program being debugged hits a breakpoint, step, and so on.

To demonstrate how these APIs are used, this topic presents an example debugger with complete code examples and an explanation of what the APIs do.

The ILE debugger that comes with IBM i uses the debugger APIs just as a user-written debugger would. There is nothing special about the IBM i debugger. Its functions could be done by an application developer using the debugger APIs and other IBM i APIs.

Scenario: A simple debugger

Consider a simple scenario in which the user wishes to debug an ILE program.

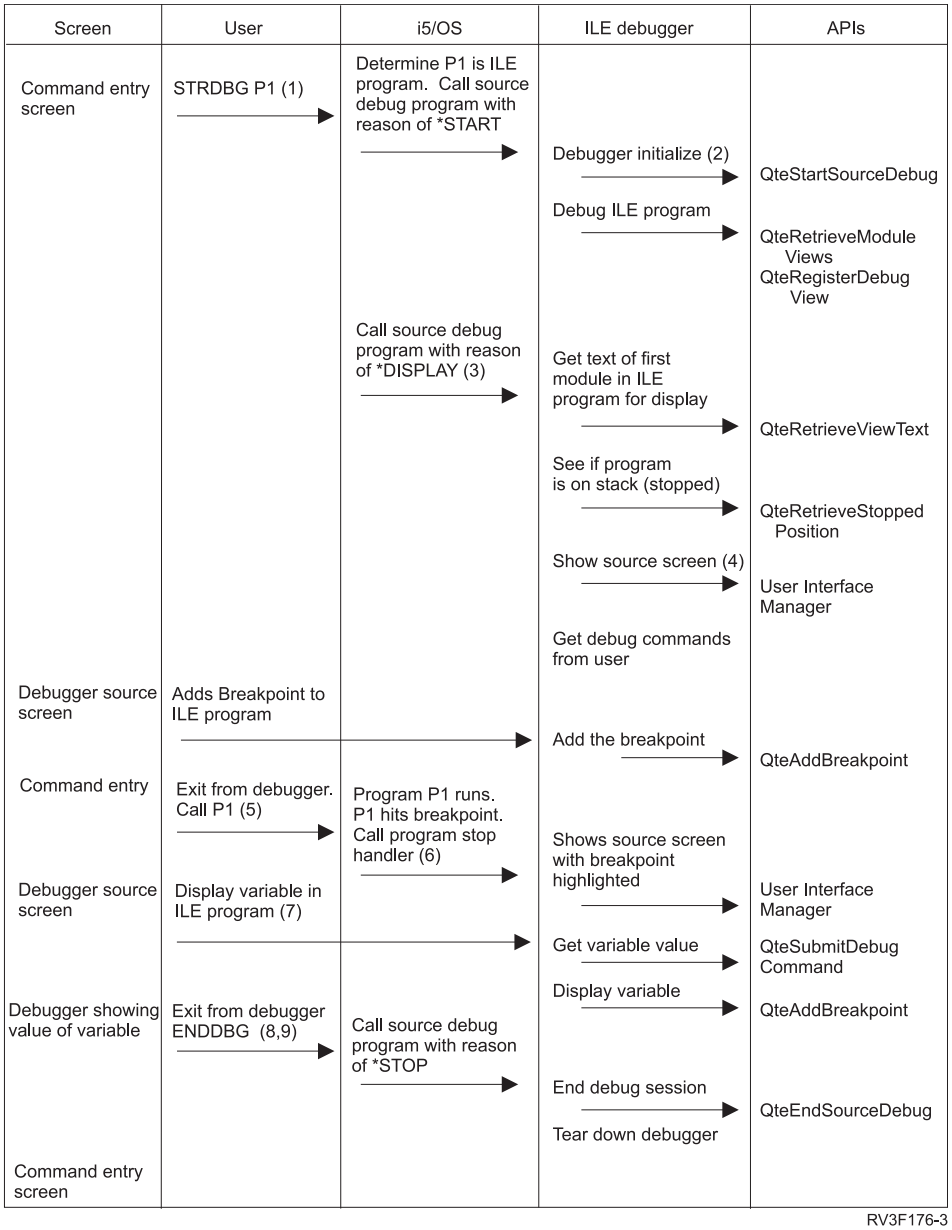
1. From the command entry screen, the user enters the Start Debug (STRDBG) command, passing it the name of an ILE program to debug.
   ```
   STRDBG P1
   ```
2. The ILE debugger screen is displayed, showing the source of a module in the ILE program being debugged. From this screen, the user adds a breakpoint and then exits.
3. Back at the command entry screen, the user runs the ILE program that is being debugged.
   ```
   CALL P1
   ```

4. The ILE program hits the breakpoint previously set. The ILE debugger screen is displayed, highlighting in the source where the program has stopped at the breakpoint.
5. The user displays a variable in the program being debugged.
6. The user exits the ILE debugger, allowing the ILE program to run to completion. The program ends.
7. Back at the command entry screen, the user ends the debug session.

   ```
   ENDDBG
   ```

This is the simplest of debug scenarios, but it illustrates how IBM i, the debugger user exits, and the debugger APIs interact.

The following figure shows the various interactions.



RV3F176-3

A detailed explanation of the scenario follows:

1. The Start Debug (STRDBG) CL command is used to start the debug session. By default, if an ILE program is specified on the command, the IBM i ILE debugger user exit is called. A different user exit (called the Source Debug program) can be specified on the Start Debug command by specifying a program name on the SRCDBGPGM parameter.

   When the Source Debug program is called, it is passed a reason field, which indicates why it was called. The *START reason is passed to it by the Start Debug command, indicating that the ILE debugger is to start itself and do any necessary initialization. When the *START reason is indicated, the names of any ILE programs on the Start Debug command are also passed to the Source Debug program.
2. In this scenario, the system Source Debug program initializes itself. It calls the QteStartSourceDebug API, which tells the system that ILE debugging is to be done. The name of a program stop handler program is passed to this API. The stop handler is a program that the system calls when an ILE program hits a breakpoint, step, or other condition where the system stops the program for the debugger.

   The Source Debug program must indicate to the system that the ILE programs specified on the Start Debug command are to be debugged. To do this, the QteRetrieveModuleViews API is called, once for each ILE program specified on the Start Debug command. In this scenario, the API is called, passing it the name of program P1. The purpose of the API is to return information about the ILE program, including the modules and views of the program. A view is the source text that is displayed by the debugger for a particular module.

   Once information about the ILE program is obtained, one or more views of the program must be registered. Once a view is registered, the system can perform various functions on that view in behalf of the debugger application. For performance reasons, only the views the user is interested in displaying should be registered.

   The Source Debug program is now done performing the function for the *START reason. It exits, returning control to the Start Debug command.
3. By default, if an ILE program is specified on the Start Debug command, the ILE debug screen is displayed. To indicate to the ILE debugger that a screen is to be put up, the Source Debug program is called by the command again, this time with a reason of *DISPLAY.

   Because this is the first time any views for P1 are to be displayed, the ILE debugger must retrieve the text to display. The first view of the first module of the program is selected as the default view to display.

   The Source Debug program calls the QteRetrieveViewText API to retrieve the text associated with the default view. Next, in case this program is already on the stack and stopped, the QteRetrieveStoppedPosition API is called to check. If the program were on the stack, the source would be positioned to the statement where the program was stopped, and that line would be highlighted. In this scenario, the program is not yet on the stack, so the first line of the source will appear at the top of the screen, and no line will be highlighted.

   The Source Debug program next calls User Interface Manager (UIM) APIs to display the source on the screen.
4. At this point, the source screen is displayed showing the text of the first view in the first module of the first ILE program specified on the Start Debug command. From this screen, the user can enter debug commands or do other options provided by the debugger application.

   In this scenario, the user adds a breakpoint to a line in the ILE program P1 being debugged. When a command is entered, the UIM APIs call a program which is part of the ILE debugger to process the command.

   To process the breakpoint, the QteAddBreakpoint is called. It is passed a view number which indicates the view being displayed, and a line number in that view. A breakpoint is added to the program by the API.
5. Back to the UIM screen, the user exits the ILE debugger. Once at the command entry screen, the user then runs the program P1 which has the breakpoint.

6. When P1 hits the breakpoint, the system calls the program stop handler defined by the QteStartSourceDebug API. The Program Stop Handler calls UIM to put up the source for the module where the program has stopped because of the breakpoint. The line is highlighted to show the user exactly where the program has stopped.
7. From the source debugger screen, the user displays a variable in program P1 which is stopped at the breakpoint. UIM calls the debugger to process the command. The debugger calls the QteSubmitDebugCommand API, which retrieves the value of the variable to be displayed. The debugger then displays this value on the screen.
8. The user now exits from the source debugger screen. This allows P1, which was stopped at a breakpoint, to continue running. When P1 ends, the user is back at the command entry screen.
9. The user ends the debug session by entering the End Debug (ENDDBG) CL command. The system calls the Source Debug program, passing it a reason of *STOP. The Source Debug program calls the QteEndSourceDebug API to indicate to the system that ILE debugging has ended. It then tears down its own environment (closes files, frees space, and so on) and then ends. The End Debug command completes, and the user is back to the command entry, the debug session having ended.

Example: Source debugger

This section discusses an example ILE debugger that demonstrates the use of some of the ILE debugger APIs. Each function in the C program is discussed along with the APIs that they call. Although the entire program listing is printed later (see "Debugger code sample" on page 443), each function or piece of code is printed with the section where it is discussed to make reading the code easier.

The example debugger does not use all ILE debugger APIs. Its function is limited. After the discussion of the code, the APIs and some functions not covered are discussed.

Compiling the debugger

The Create C Module (CRTCMOD) command compiles the source code of the debugger. It is compiled into module DEBUG.

The Create Program (CRTPGM) command creates program DEBUG from module DEBUG. It is necessary to bind to service program QTEDBGS so that the calls to the debugger APIs are resolved. It is also important to use activation group QTEDBGAG. This is an activation group that cannot be destroyed while the job is in debug mode. Thus, all static variables in program DEBUG remain intact throughout the debugging of the ILE program. Only when ENDDBG is entered can the activation group be destroyed, even if the Reclaim Resources (RCLRSC) CL command is entered.

Starting the debugger

The example debugger consists of a single program called DEBUG. The program is used as the Source Debug program as well as the Program Stop Handler. The program determines how many parameters it is being called with, and with this information it does the function of one or the other of the user exits.

The debugger can debug only one ILE program. This program is specified on the Start Debug CL command. The program cannot be removed from debug until ENDDBG is done. No new programs can be added.

To debug an ILE program P1 with this sample debugger, the following CL command could be entered:

```
STRDBG P1 SRCDBGPGM(DEBUG)
```

Note that DEBUG must be in the library list when STRDBG is done.

If the command is done, P1 is called twice, once as a Source Debug program given a reason of *START, and again as a Source Debug program given a reason of *DISPLAY.

Other variations of the Start Debug command can be given with different results. For example, the following CL command causes DEBUG to be called only once with a reason of *START:

```
STRDBG P1 SRCDBGPGM(DEBUG) DSPMODSRC(*NO)
```

This is because STRDBG has been told not to display the debug screen, so the *DISPLAY reason is not given until the user does the Display Module Source (DSPMODSRC) CL command.

The following example does not even call DEBUGGER:

```
STRDBG SRCDBGPGM(DEBUG)
```

This is because no ILE program is specified. If an ILE program receives an unmonitored message and the ILE debugger needs to be called, DEBUG is first called with *START as a Source Debug program. Also, if Display Module Source is entered, the *START and then the *DISPLAY reason is passed to DEBUG.

Using the debugger

When the debugger is started, it allows simple debugging commands to be entered. The C session manager is put up, which scrolls the users commands and the debugger output. To see a list of the allowable commands, enter `HELP`.

The "list views" command shows all of the views available in the program being debugged. The text description of the view is listed, with a sequential number. This number is used by the "switch" command to switch to that view.

The "list text" command prints out the text of the current view. Text has a line number next to it. The line number is used when setting breakpoints or other debug commands.

The switch command switches the current view. The current view is the view used when setting breakpoints, displaying variables, viewing text, and so on.

The "quit" command exits the debugger.

Other commands are interpreted by the QteSubmitDebugCommand API. This API will be discussed later. An example command that can be entered is "break n", where n is the line number in the current view. These commands are similar to the ones allowed in the ILE debugger shipped with IBM i.

Header files used in debugger

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <qtedbgs.h>
```

Besides the normal C library header files, an API header file, qtedbgs.h is included. This file defines the functions exported from service program QTEDBGS. This service program contains the ILE debugger APIs.

Global variables

```
static _TE_VEWL0100_T *pgm_dbg_dta = NULL;
static long current_view = 0;       /* current view - defaults to 1st*/
static _TE_OBJLIB_T program_lib;    /* name and lib of pgm debugged  */
```

These are global variables that hold information about the program being debugged. These variables do not go away when program DEBUG exits, because they are stored in the activation group which is not destroyed until the debug session has completed.

The name and library of the program are stored, as is the current view being debugged. Also, a pointer to a structure returned by the QteRetrieveModuleViews is saved, as this information is needed when debugging the various views of the program.

PgmList_t

```
typedef struct {
    _TE_OBJLIB_T PgmLib;            /* Name and Library of program   */
    _TE_NAME_T PgmType;             /* program type, *PGM or *SRVPGM */
} PgmList_t;
```

This is the structure of the name, library, and type of the program being debugged.

main()

```
main (int argc, char *argv[]) {
  if (argc == 4)                    /* called as source debug program*/
    HandleSession(argv[1], (PgmList_t *)argv[2], *(int
*)argv[3]);
  else if (argc == 8)               /* called as program stop handler*/
    HandleStop((_TE_OBJLIB_T *)argv[1], argv[2],
argv[3], argv[4],
               (long *)argv[5], *(int *)argv[6],
argv[7]);
}
```

Program DEBUG can be called in two ways. When it is called by the STRDBG, DSPMODSRC, and ENDDBG CL commands, it is called as the Source Debug program user exit. It is passed three parameters.

DEBUG can also be called when a program being debugged hits a breakpoint or step. In this case, it is passed seven parameters.

DEBUG therefore can determine why it was called by counting the number of parameters it was passed. Remember that argc includes the program name as the first argument passed.

If argc is 4 (three parameters passed to DEBUG), function HandleSession is called, and the three parameters passed to DEBUG are passed to it, typecasted as needed.

If argc is 8 (seven parameters passed to DEBUG), function HandleStop is called, and the seven parameters passed to DEBUG are passed to it, typecasted as needed.

If any other number of parameters are passed to DEBUG, it cannot have been called from the IBM i debug support, so DEBUG will just exit.

HandleSession()

```
void HandleSession(char reason[10],
                   PgmList_t ProgramList[],
                   int ProgramListCount) {

  if (memcmp(reason,"*START    ",10) == 0) /* reason is *START       */
    StartUpDebugger(ProgramList, ProgramListCount);
  else if ( memcmp(reason,"*STOP     ",10) == 0) /* reason is *STOP  */
    TearDownDebugger();
  else if ( memcmp(reason,"*DISPLAY  ",10) == 0) /* reason *DISPLAY  */
    ProcessCommands();
}
```

When DEBUG is called as a session handler, it is passed three parameters. The first parameter is a 10-character array containing a reason field. This contains the reason why the session handler is called.

When DEBUG is first called, it is passed a reason of *START, indicating that the debugger is to initialize for an ILE debug session. When this reason is given, the second parameter contains a list of ILE programs specified on the STRDBG command, and the third parameter contains the number of programs specified on parameter two. From 0 to 10 ILE programs can be specified.

When the user wishes to see the ILE debugger screen, either from STRDBG or DSPMODSRC, a reason of *DISPLAY is passed. When the user enters ENDDBG, the *STOP reason is passed, indicating that the ILE debug session is ending. The second and third parameters are not used when the reason is *DISPLAY or *STOP.

The code tests for a reason and calls the appropriate function. There is one function for each reason that can be passed.

TearDownDebugger()

```
void TearDownDebugger(void) {
  _TE_ERROR_CODE_T errorCode = {8}; /* errors will be ignored        */

  /* Call EndSourceDebug to get out of ILE debug mode               */
  QteEndSourceDebug(&errorCode);

  exit(0);                          /* destroy activation group      */
}
```

This function is called when the user enters ENDDBG. The debugger calls the QteEndSourceDebug API which ends ILE debugging. Since an 8 is passed as the number of bytes provided, the message ID and error data from an error are not returned to the caller. Thus, any errors from this API (there should not be any) are ignored.

The exit() function is called, which destroys the activation group. Thus, all global data defined in the program's variables are lost. This is ok, since the debug session is ending at this point.

StartUpDebugger()

```
void StartUpDebugger(PgmList_t ProgramList[],
                     int ProgramListCount) {

  _TE_ERROR_CODE_T errorCode = {0}; /* exceptions are generated      */
  _TE_OBJLIB_T StopHandler = {"DEBUG     ", "*LIBL     "};
  int i;

  if (ProgramListCount!=1) {        /* is only 1 pgm passed on STRDBG*/
    printf("Exactly ONE program must be specified on STRDBG.\n");
    TearDownDebugger();             /* end debugger                  */
  }

  /* Copy program name to global variables                          */
  memcpy(&program_lib, &ProgramList->PgmLib, 20);

  /* Call StartSourceDebug: giving the name and library of the      */
  /* stop handler.  This will start ILE debug mode                  */
  QteStartSourceDebug(&StopHandler, &errorCode);

  AddProgram();                     /* add program to debug          */
}
```

This function is passed the second and third parameters which were passed from the system when it called DEBUG with a reason of *START. These parameters are the list of programs to be added to debug and the number of programs in the list. This simple example debugger can only debug one program, so if any other number of programs were specified on STRDBG, the debugger just exits.

StartUpDebugger first stores the program/library element passed to it in a global variable available to all functions. This is the name and library of the program being debugged. It then calls the QteStartSourceDebug API to tell the system that an ILE debug session is to begin. The name and library of program DEBUG are passed to this API as the Program Stop Handler. Thus, whenever the program being debugged is stopped by the debugger, program DEBUG will be called.

Finally, the function calls AddProgram to add the single program to debug.

AddProgram()

```
void AddProgram(void) {

  _TE_ERROR_CODE_T errorCode = {0};    /* Signal exceptions on error */
  _TE_NAME_T       Library;            /* Lib returned               */
  _TE_TIMESTAMP_T  TimeStamp;          /* TimeStamp returned         */
  int viewIndex;
  long int         iViewID;
  long int         iViewLines;
  long rtvModViewDataLength = 8;       /* size of receiver buffer    */
  char tempBuffer[8];        /* enough room for header only*/
  int i, tempModuleCount;

  /* Call QteRetrieveModuleViews to determine the number of bytes    */
  /* the receiver variable needs to be to hold all of the views for  */
  /* the program.                                                    */
  pgm_dbg_dta = (_TE_VEWL0100_T *)tempBuffer;
  QteRetrieveModuleViews((char *)pgm_dbg_dta, &rtvModViewDataLength,
                         "VEWL0100", &program_lib,
                         "*PGM      ", "*ALL      ", Library,
                         &errorCode);

  /* Get a buffer large enough to hold all view information          */
  rtvModViewDataLength = pgm_dbg_dta->BytesAvailable;
  pgm_dbg_dta = (_TE_VEWL0100_T *)malloc(rtvModViewDataLength);

  /* Call QteRetrieveModuleViews again, passing a big enough buffer. */
  QteRetrieveModuleViews((char *)pgm_dbg_dta, &rtvModViewDataLength,
                         "VEWL0100", &program_lib,
                         "*PGM      ", "*ALL      ", Library,
                         &errorCode);

  /* If number of elements is zero, program is not debuggable.       */
  if (pgm_dbg_dta->NumberElements == 0) {
    printf("Program %.10s in Library %.10s cannot be debugged.",
           program_lib.obj, program_lib.lib);
    TearDownDebugger();
  }

  /* Put the library returned by Retrieve Module Views in PgmLib     */
  memcpy(program_lib.lib, Library, sizeof(_TE_NAME_T));

  /* Register all views in the program                               */
  for (i=0; i < pgm_dbg_dta->NumberElements; i++) {
    QteRegisterDebugView(&iViewID, &iViewLines, Library, TimeStamp,
                         &program_lib, "*PGM      ",
                         pgm_dbg_dta->Element[i].ModuleName,
                         &pgm_dbg_dta->Element[i].ViewNumber,
                         &errorCode);

    /* overwrite unneeded ViewNumber with obtained view id           */
    pgm_dbg_dta->Element[i].ViewNumber = iViewID;
  }
}
```

The heart of this function is the two calls to the QteRetrieveModuleViews API and the call to QteRegisterDebugView API.

The QteRetrieveModuleViews API returns information about an ILE program. It returns this information in a structure of type _TE_VEWL0100_T. This is a fairly complex structure that has the following fields:

```
typedef _Packed struct {            /* format VEWL0100              */
  long int BytesReturned;           /* number of bytes returned     */
  long int BytesAvailable;          /* number of bytes available    */
  long int NumberElements;          /* number of elements returned  */
  _Packed struct {                  /* one element                  */
    _TE_NAME_T ModuleName;          /* name of module in program    */
    _TE_NAME_T ViewType;            /* type of view:                */
    _TE_COMPILER_ID_T CompilerID;   /* compiler ID                  */
    _TE_NAME_T MainIndicator;       /* main indicator               */
    _TE_TIMESTAMP_T TimeStamp;      /* time view was created        */
    _TE_TEXTDESC_T ViewDescription; /* view description             */
    char Reserved[3];
    long int ViewNumber;            /* view number within module    */
    long int NumViews;              /* number of views in this module*/
  } Element[1];                     /* one element         */
} _TE_VEWL0100_T;
```

This structure has a header portion which holds the number of bytes returned by the API (BytesReturned), the number of bytes that can be returned by the API, used when there is not enough room for the API to return all of its data (BytesAvailable), and the number of elements (views) returned by the API (NumberElements).

Since there is no way to know in advance how many views a program has, the QteRetrieveModuleViews API should be called once with only enough storage to return the number of bytes that the API needs to return all of its information. Thus, the first call to the API provides only 8 bytes of storage for the API to return its data. This allows the API to fill in the BytesAvailable field.

QteRetrieveModuleViews is passed a buffer to hold the receiver variable and the length of that buffer (in this case, 8 bytes). It is also passed a format name which identifies the structure of the receiver variable. The only allowable format name at this time is VEWL0100. A structure containing the program name and library name of the ILE program is passed. Also, the program type is passed. In this example debugger, only *PGM objects can be debugged, but it is possible to debug *SRVPGM objects using the ILE debugger APIs.

The name of the module is provided, in which case information about that module is returned. *ALL indicates that information about all modules in the program is to be returned. A return library variable is passed. This is so that when *LIBL is passed as a library name, the real library name can be obtained, making subsequent API calls faster because the library list won't have to be searched again.

Finally an error code structure is passed to the API. This structure is initialized with a zero, indicating that the API is not to fill in any error code data. Instead, the API issues an exception if an error occurs. No errors are expected, so this should not matter.

Before QteRetrieveModuleViews is called again, a buffer large enough to hold all of the information is created. The API is called again with the same parameters, but this time the entire information will be stored by the API in the allocated buffer.

If the API does not return any elements, this means that none of the modules has debug data. In this case, the program cannot be debugged, so the debug session is ended.

Now that a list of views has been retrieved, it is time to register all of the views to the system, making it possible to do debug operations against them. In a real debugger, only the views requested to be seen by the user would be registered to save processing time, but in this example, all views will be registered at once.

Not all of the fields in the VEWL0100 structure are needed by this debugger. However, they are described here. The API returns one element for each view in the program. Each module in the program might have several views. All views for a particular module are contiguous in the list.

View	Description
ModuleName	This is the name of the module in the program which this particular view is for.
ViewType	This indicates the type of view. A *TEXT view contains the text retrieved from source files on the system. The text contains sequence information from these files that the debugger might not want to display. A *LISTING view contains the text that is stored with the program object itself. A *STATEMENT view contains the information about HLL statements in the module, and this information is not generally displayed to the user but is used by the debugger. In the case of this debugger, all views are displayed exactly as the text for the views is retrieved.
CompilerID	This indicates the language that the particular module is written in. This is not used by the example debugger.
MainIndicator	Only one module in a program is the module with the program entry procedure (main() in the case of ILE C programs). If a particular view in the list comes from this module, then this field indicates that the module contains this procedure. This field is not used by the example debugger.
TimeStamp	This indicates when the view was created. This is useful in allowing the debugger to detect if a program has been recompiled and the debugger has down-level view information. This field is not used by the example debugger.
ViewDescription	This is text given to the view by the compiler creating the view. It is a description of the view which can be displayed by the debugger.
ViewNumber	This is a sequence number of the view in a particular module. When registering a view, the program name, module name, and view number must be provided.
NumViews	This is how many views are in the module. All elements for views in a given module have the same value for this field. This field is not used by the example debugger.

A loop through all the views returned by QteRetrieveModuleViews is done, registering the view using the QteRegisterDebugView API. The program name, program type, module name, and view number of the module are passed as inputs to the API. The API returns the library of the program (in case *LIBL) is passed in as the program library), the timestamp of the view (in case the program has been recompiled between the time the view information was retrieved and the time the view was registered), the number of lines of text in the view, and a view ID. The view ID is a handle, and it is used in identifying the registered view to various APIs. For example, when retrieving text for a particular view, the view must be registered, and the view ID returned when registering the view is passed to the QteRetrieveViewText API.

The structure that held the views retrieved by QteRetrieveModuleViews is also used by the debugger. The view number is no longer needed, since it is just a sequence number passed to QteRegisterDebugView. Thus, this number is overwritten and will hold the view ID, which is needed by other debugger APIs.

ProcessCommands()

```
void ProcessCommands(void) {
  char InputBuffer[80];
  char *token;
  int i;
  int step=0;                       /* do an exit for step when 1    */

  if (pgm_dbg_dta == NULL) {        /* if no debug data              */
    printf("Debug session has ended.\n");
    exit(0);                        /* end the debugger              */
  }

  while(!step) {                    /* read until step or quit cmd   */
    ReadLine(InputBuffer,sizeof(InputBuffer));
    token = strtok(InputBuffer," ");

    if (token==NULL) continue;      /* ignore blank lines            */
    else if (strcmp(token,"quit") == 0) /* the quit command?         */
      return;                       /* exit debugger                 */
    else if (strcmp(token,"list") == 0) /* the list command?         */
      ProcessListCommand();         /* process command               */
    else if (strcmp(token,"switch") == 0) { /* switch command?       */
      token = strtok(NULL," ");     /* get view number token         */
      if (token == NULL)
        printf("'switch' must be followed by a view number.\n");
      else
        current_view = atoi(token);   /* switch current view         */
    }
    else if (strcmp(token,"help") == 0) {
      printf("The following are the allowed debugger commands:\n");
      printf("  list views - lists all views in the program\n");
      printf("  list text - lists the text of the current view\n");
      printf("  switch n - changes current view to view n\n");
      printf("  help - displays this help text\n");
      printf("  quit - ends the debug session\n");
      printf("Other commands are interpreted by the debug support.\n");
    }
    else {                          /* pass command to API           */
      /* Undo modifications that strtok did                          */
      InputBuffer[strlen(InputBuffer)] = ' ';

      step = ProcessDbgCommand(InputBuffer);
    }
  }
}
```

This function reads an input line from the user and processes it. If it is a command recognized by the debugger, it process it. If not, it calls ProcessDebugCommand which lets QteSubmitDebugCommand process the command.

The first test is to make sure that the pointer to the debug data is not null. This is here for safety reasons. If program DEBUG is compiled with the wrong activation group name or no name at all, its global variables can be destroyed when the program exits, causing problems when the program is called again. This test prevents debug commands from being entered if the activation group has been destroyed, wiping out the global view data.

The function loops until the quit command is entered or until a step is done. It calls the appropriate function based on the command entered, or displays an error message if a syntax error is detected. If the command is unknown, it is processed by ProcessDbgCommand.

The switch command is processed directly by the function. It changes the current view to a number provided. There is no error checking in this sample debugger.

ReadLine()

```
void ReadLine(char *Buffer, int length) {
  int i;                            /* loop counter                 */

  printf("Enter a debugger command or 'help'.\n");
  fgets(Buffer,length,stdin);       /* read line of text            */

  /* Blank out line from \n to the end of the string.               */
  for (i=0; i<length; i++) {        /* loop, searching for newline  */
    if (Buffer[i] == '\n') {        /* if newline character found
*/
      break;                        /* end loop searching for newline*/
    }
  }

  memset(Buffer+i,' ',length-i);     /* blank remainder of line     */
}
```

This function reads a line of text from the user and fills the input buffer with trailing blanks.

ProcessListCommand()

```
void ProcessListCommand(void) {
  char *token;                      /* pointer to next token of input*/

  token = strtok(NULL," ");         /* get next token in input buffer*/

  if (token==NULL)                  /* list not followed by anything */
    printf("'list' must be followed by 'views' or 'text'.\n");
  else if (strcmp(token,"views") == 0)/* if list views               */
    PrintViews();
  else if (strcmp(token,"text") == 0) /* if list text                */
    PrintText();
  else                              /* list <something-else>        */
    printf("'list' must be followed by 'views' or 'text'.\n");
}
```

This routine process the list command. There are two versions of the list command, list views and list text. The appropriate function is called depending on the type of list command entered, or a syntax error message is issued.

PrintViews

```
void PrintViews(void) {
  int k;

  /* loop through views printing view#, module, and view desc. text  */
  for (k=0; k< pgm_dbg_dta->NumberElements; k++) {
    printf("%d) %.10s:%.50s",
           k,
           pgm_dbg_dta->Element[k].ModuleName,
           pgm_dbg_dta->Element[k].ViewDescription);
    if (current_view == k)          /* indicate if view is current   */
      printf("<---Current\n");
```

```
    else
      printf("\n");
  }
}
```

This routine lists all of the views available in the program being debugged. The information about the views is stored in the buffer that was passed to QteRetrieveModuleViews.

The module name and view descriptive text is printed for each view. If the current view being printed is also the current view, this is noted by printing this fact next to the view information.

A view number is printed next to each view. This is not the view ID returned by the QteRegisterDebugView. It is a number allowing the user to change the current view to one of the views in the list.

PrintText()

```
void PrintText(void) {

  long LineLength = 92;             /* length of lines of text       */
  long NumberOfLines = 0;           /* lines to retrieve - 0 = all   */
  long StartLine=1;                 /* retrieve from line 1 (first)  */
  long bufferLength = 100000;       /* size of retrieved text buffer */
  long viewID;                      /* view ID of text to retrieve   */
  _TE_TEXT_BUFFER_T *buffer;        /* text retrieved by API         */
  _TE_ERROR_CODE_T errorCode = {0}; /* Exceptions will be signaled   */
  int i;                            /* points to start of each line  */
  int line_number;                  /* line number counter for loop  */

  /* Get View ID of current view                                     */
  viewID = pgm_dbg_dta->Element[current_view].ViewNumber;

  buffer = malloc(bufferLength);    /* malloc space for big text buf */

  /* Call Retrieve_View_Text for the current view.                   */
  QteRetrieveViewText((char *)buffer, &bufferLength, &viewID,
                      &StartLine, &NumberOfLines, &LineLength,
                      &errorCode);

  /* Print out the text                                              */
  for (i=0,line_number=1;
       line_number <= buffer->NumLines;
       line_number++,i+=LineLength) {
     printf("%3d) %.70s\n", line_number, buffer->Text+i);
  }

  free(buffer);                     /* free memory for buffer        */
}
```

This function retrieves the text associated with the current view and prints it. This text is the source of the program and is the heart of a source debugger screen.

The text of the current view is retrieved, so the view ID of that view is determined. It is this view that is passed to QteRetrieveViewText.

In the sample debugger, a large buffer is allocated, and as much text as will fit in this buffer is retrieved. The QteRetrieveViewText API returns the text and the number of lines that fit in the buffer.

Once the text is retrieved, it is printed out along with the line number. The line number is needed when setting breakpoints based on the view.

ProcessDbgCommand()

```
int ProcessDbgCommand(char InputBuffer[80]) {
  _TE_ERROR_CODE_T errorCode = {64}; /* fill in bytes provided       */
  char OutputBuffer[4096];
  struct _TE_RESULT_BUFFER_T *Results;
  long InputBufferLength = 80;
  long OutputBufferLength = sizeof(OutputBuffer);
  long view_ID;
  _TE_COMPILER_ID_T *CompilerID;
  int i;
  int return_value = 0;

  view_ID = pgm_dbg_dta->Element[current_view].ViewNumber;
  CompilerID = &pgm_dbg_dta->Element[current_view].CompilerID;

  /* Give command to QteSubmitDebugCommand                           */
  QteSubmitDebugCommand(OutputBuffer, &OutputBufferLength,
                        &view_ID, InputBuffer, &InputBufferLength,
                        *CompilerID, &errorCode);

  if (errorCode.BytesAvailable != 0) {
    printf("Error = %.7s\n",errorCode.ExceptionID);
    return return_value;
  }

  /* Process results from QteSubmitDebugCommand                      */
  Results = (_TE_RESULT_BUFFER_T *) OutputBuffer;

  /* Loop through Results array                                      */
  for (i=0; i<Results->Header.EntryCount; i++) {
    switch (Results->Data[i].ResultKind)
    {
      case _TE_kStepR            :
        printf("Step set\n");
        return_value=1;             /* indicate step is to be done   */
        break;
      case _TE_kBreakR           :
        printf("Breakpoint set");
        break;
      case _TE_kBreakPositionR   :
        printf(" at line %d\n",
               Results->Data[i].V.BreakPosition.Line);
        break;
      case _TE_kExpressionTextR  :
        printf("%s",
               ((char *)Results) + Results->Data[i].V.
               ExpressionText.oExpressionText);
        break;
      case _TE_kExpressionValueR :
        printf(" = %s\n",
               ((char *)Results) + Results->Data[i].V.
               ExpressionValue.oExpressionValue);
        break;
      case _TE_kQualifyR         :
        printf("Qual set\n");
        break;
      case _TE_kClearBreakpointR :
        printf("Breakpoint cleared\n");
        break;
      case _TE_kClearPgmR :
        printf("All breakpoints cleared\n");
        break;
      default:                      /* ignore all other record types */
        break;
    }                               /* switch                        */
  }                                 /* loop through results array    */
  return return_value;
}
```

This function is called to process all commands not known by the debugger. It calls the QteSubmitDebugCommand API which is passed a view ID, compiler ID, and a command. The API needs the compiler ID because each programming language used in compiling a particular module has different debug commands or command syntax, and the API needs to know which language was used when compiling the module.

The API returns back a series of result records which indicate what was done by the API. Most of this function reads the results of the records returned and prints an appropriate response message.

Some results records indicate that a particular function has been performed. These include:

Result record	Description
_TE_kStepR	The step command was successfully done.
_TE_kBreakR	The break command was successfully done.
_TE_kQualifyR	The qual command was successfully done.
_TE_kClearBreakpointR	The clear breakpoint command was successfully done.
_TE_kClearPgmR	The clear pgm command was successfully done.

Other results records contain numeric data useful by the debugger.

Result record	Description
_TE_kBreakPositionR	Contains the line number where a breakpoint was set. It is possible that a breakpoint set on two different lines will correspond to the same HLL statement. In this case, only one breakpoint is really set. To determine if this is the case, it is necessary to map the position in the view where the breakpoint is set to a position in the statement view.

Still other results records contain string data. In this case, the record contains an offset into the string space returned by the API as well as a string length.

Result record	Description
_TE_kExpressionTextR	This points to the expression entered in the eval command.
_TE_kExpressionValueR	This points to the value of the evaluated expression.

There are other kinds of results records than processed by the sample debugger. The QteSubmitDebugCommand API discusses in detail each result record and the data it contains.

The API description also discusses the syntax of the debug command that must be passed to it. The commands and their syntax will not be discussed in depth here, but a few example commands will be shown:

- break 5 when x == 3

  This is a conditional breakpoint. The debugger will stop the program indicated by the view ID passed to the API when it reaches line 5 of the view and when the expression "x == 3" is true. The "when" part of the break statement is optional, in which case an unconditional breakpoint is set.
- step 1 into

  The step command instructs the debug support to stop the a program when it has executed one or more statements. In this example, the program is stopped after 1 statement has been executed. The

"into" means that statements in procedures are counted when stepping. "over" means that statements in called procedures are skipped over and not counted. The default step type is "into", and the default step count is 1.

- qual 13

  The qual command is necessary when there are blocks of code with the same variable name. In this case, the user indicates where the variable is searched for in the program. Normally, this command is not used.

- clear 8

  A conditional or unconditional breakpoint is removed from line 8 of the view indicated by the view ID parameter.

HandleStop()

```
void HandleStop(_TE_OBJLIB_T *ProgramLib,
                _TE_NAME_T ProgramType,
                _TE_NAME_T Module,
                char reason[10],
                long Statements[],
                int StatementsCount,
                char *message) {
  int i;
  _TE_MAPP0100_T Map_Return_Structure;
  long Column = 1;
  long MapLength = sizeof(Map_Return_Structure);
  _TE_ERROR_CODE_T errorCode = {64};
  long stmt_view;

  /* If current view is for a different module than the one that is  */
  /* stopped, change current view to first view in the stopped module*/
  if (memcmp(Module,
             pgm_dbg_dta->Element[current_view].ModuleName,
             sizeof(_TE_NAME_T)) != 0) { /* a different module?      */
    for (i=0; i<pgm_dbg_dta->NumberElements; i++) {
      if (memcmp(Module,
                 pgm_dbg_dta->Element[i].ModuleName,
                 sizeof(_TE_NAME_T)) == 0) { /* found module         */
        current_view = i;           /* change current view to module */
        printf("Current view changed to %d.\n",current_view);
        break;                      /* exit search loop              */
      }                             /* module found                  */
    }                               /* loop through views            */
  }                                 /* current view to be changed    */

  /* Get number of statement view for module stopped                 */
  for (i=0; i<pgm_dbg_dta->NumberElements; i++) {
    if ((memcmp(Module,
                pgm_dbg_dta->Element[i].ModuleName,
                sizeof(_TE_NAME_T)) == 0) &&
        (memcmp("*STATEMENT",
                 pgm_dbg_dta->Element[i].ViewType,
                 sizeof(_TE_NAME_T)) == 0))
      stmt_view = i;
  }


  /* Call QteMapViewPosition to map the stopped location (which      */
  /* is in terms of the *STATEMENT view) to the current view of      */
  /* the module                                                      */
  QteMapViewPosition((char *)&Map_Return_Structure, &MapLength,
                     &pgm_dbg_dta-> Element[stmt_view].ViewNumber,
                     &Statements[0], &Column,
                     &pgm_dbg_dta->Element[current_view].ViewNumber,
                     &errorCode);
```

```
  /* Tell the user about the program that stopped.                  */
  for (i=0;i<4;i++) {               /* See why program stopped      */
    if (reason[i] == '1') {
      switch(i) {
        case 0: printf("Unmonitored exception");
          break;
        case 1: printf("Breakpoint");
          break;
        case 2: printf("Step completed");
          break;
        case 3: printf("Breakpoint condition error");
          break;
      }
    }
  }
  printf(" in module %.10s at line %d.\n",
         Module,
         Map_Return_Structure.MapElem[0].LineNumber);

  ProcessCommands();                /* put user into debugger       */
}
```

This function is called when program DEBUG is called as a Program Stop Handler. It is passed the name, library, and type of the program stopped, the line number in the statement view where it has stopped, a count of line numbers stopped in, if the system cannot determine exactly where the program has stopped (this is the case for optimized code), and an array of character flags indicating why the program was stopped.

The first thing the function does is determine if the current view is set to the module where the program stopped. If not, then it needs to be reset to the first view in the module where the program has stopped.

Next, the statement view ID for the module stopped needs to be determined. This is necessary because the stopped position is given in terms of the statement view, and this position needs to be converted to a position in the current view.

The QteMapViewPosition API maps a position in the statement view to a statement in another view in that module. This allows the debugger to determine the source line of the current view where the program has stopped, even though the program is only told the line number in the statement view.

Finally, the character flags are checked to see why the program was stopped. Note that the program can be stopped for more than one reason, so every flag is checked, and if it is on, a message for that flag is printed.

Finally, the ProcessCommands function is called, allowing the user to enter debug commands.

Other APIs

This section discusses other APIs not covered in this example debugger. Some or all of these APIs could be used in a real ILE source-level debugger. All of them are used in the debugger shipped with IBM i.

QteRetrieveDebugAttributes

This API allows a debugger to retrieve information about the debug session. This includes the value of the Update Production Files, set on the Start Debug command, as well as an indication of whether the job where the debugger is running is servicing and debugging another job.

QteSetDebugAttributes

The only attribute that can be set is the value of the Update Production Files. This can also be accomplished using the Change Debug (CHGDBG) CL command.

QteRemoveDebugView

Views that are registered can be removed from debug. This is desirable if a program is to be removed from debug so that it can be recompiled and added again. It is not necessary to remove views from debug when ending the debug session, as QteEndSourceDebug will do this automatically.

QteRetrieveStoppedPosition

This indicates if a program is currently stopped and on the stack, and whether this stopped position is anywhere in a given view. This is useful whenever a source debugger is about to put up a source screen. If the program is stopped somewhere within the source to be displayed, this can be indicated to the user.

This is necessary because a program can be stopped by other means than the debugger. For example, an ILE program could have put up a command entry screen, and the debugger could be displayed from there. In this case, it is nice to indicate to the user that the program being debugged is stopped.

QteAddBreakpoint

This and the following APIs are not really needed, as their function can be done with the QteSubmitDebugCommand. However, this API is much faster, since a debug language command does not need to be parsed and interpreted. In cases where the debugger knows the information without needing to specify a debug command to the API, these "shortcut" APIs should be used.

This API performs the same function as the `break n` debug language command.

QteRemoveBreakpoint

This API performs the same function as the `clear n` debug language command.

QteRemoveAllBreakpoints

This API performs the same function as the `clear pgm` debug language command.

QteStep

This API performs the same function as the `step n into` and `step n over` debug language commands.

Debugger code sample

Here is the entire program listing for the ILE C program that contains the example debugger:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  The entire program listing for the program             */
/*            containing the example debugger discussed in the       */
/*            preceding sections.                                    */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: QteRetrieveViewText, QteSubmitDebugCommand,            */
/*            QteEndSourceDebug, QteRetrieveModuleViews,             */
/*            QteRegisterDebugView, QteStartSourceDebug,             */
/*            QteMapViewPosition                                     */
/*                                                                   */
/*********************************************************************/
```

```
/*****************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <qtedbgs.h>

/* Global variables holding information about a program in debug mode*/
static _TE_VEWL0100_T *pgm_dbg_dta = NULL;
static long current_view = 0;       /* current view - defaults to 1st*/
static _TE_OBJLIB_T program_lib;    /* name and lib of pgm debugged  */

/* ReadLine: Reads a line of input and stores it in a string.        */
void ReadLine(char *Buffer, int length) {
  int i;                            /* loop counter                 */

  printf("Enter a debugger command or 'help'.\n");
  fgets(Buffer,length,stdin);       /* read line of text            */

  /* Blank out line from \n to the end of the string.               */
  for (i=0; i<length; i++) {        /* loop, searching for newline  */
    if (Buffer[i] == '\n') {        /* if newline character found */
      break;                        /* end loop searching for newline*/
    }
  }

  memset(Buffer+i,' ',length-i);     /* blank remainder of line     */
}

/* PrintText: This function will print the text for the current view */
void PrintText(void) {

  long LineLength = 92;             /* length of lines of text      */
  long NumberOfLines = 0;           /* lines to retrieve - 0 = all  */
  long StartLine=1;                 /* retrieve from line 1 (first) */
  long bufferLength = 100000;       /* size of retrieved text buffer */
  long viewID;                      /* view ID of text to retrieve  */
  _TE_TEXT_BUFFER_T *buffer;        /* text retrieved by API        */
  _TE_ERROR_CODE_T errorCode = {0}; /* Exceptions will be signaled  */
  int i;                            /* points to start of each line */
  int line_number;                  /* line number counter for loop */

  /* Get View ID of current view                                    */
  viewID = pgm_dbg_dta->Element[current_view].ViewNumber;

  buffer = malloc(bufferLength);    /* malloc space for big text buf */

  /* Call Retrieve_View_Text for the current view.                  */
  QteRetrieveViewText((char *)buffer, &bufferLength, &viewID,
                      &StartLine, &NumberOfLines, &LineLength,
                      &errorCode);

  /* Print out the text                                             */
  for (i=0,line_number=1;
       line_number <= buffer->NumLines;
       line_number++,i+=LineLength) {
     printf("%3d) %.70s\n", line_number, buffer->Text+i);
  }

  free(buffer);                     /* free memory for buffer       */
}

/* PrintViews: Prints all the views of the program being debugged.  */
void PrintViews(void) {
  int k;

  /* loop through views printing view#, module, and view desc. text  */
```

```
  for (k=0; k< pgm_dbg_dta->NumberElements; k++) {
    printf("%d) %.10s:%.50s",
           k,
           pgm_dbg_dta->Element[k].ModuleName,
           pgm_dbg_dta->Element[k].ViewDescription);
    if (current_view == k)         /* indicate if view is current   */
      printf("<---Current\n");
    else
      printf("\n");
  }
}

/* ProcessListCommand:  Process list command to list views or text   */
void ProcessListCommand(void) {
  char *token;                     /* pointer to next token of input*/

  token = strtok(NULL," ");        /* get next token in input buffer*/

  if (token==NULL)                 /* list not followed by anything */
    printf("'list' must be followed by 'views' or 'text'.\n");
  else if (strcmp(token,"views") == 0)/* if list views               */
    PrintViews();
  else if (strcmp(token,"text") == 0) /* if list text                */
    PrintText();
  else                             /* list <something-else>         */
    printf("'list' must be followed by 'views' or 'text'.\n");
}

/* ProcessDbgCommand:  This function will process commands sent to   */
/*                     the QteSubmitDebugCommand API.                */
int ProcessDbgCommand(char InputBuffer[80]) {
  _TE_ERROR_CODE_T errorCode = {64}; /* fill in bytes provided       */
  char OutputBuffer[4096];
  struct _TE_RESULT_BUFFER_T *Results;
  long InputBufferLength = 80;
  long OutputBufferLength = sizeof(OutputBuffer);
  long view_ID;
  _TE_COMPILER_ID_T *CompilerID;
  int i;
  int return_value = 0;

  view_ID = pgm_dbg_dta->Element[current_view].ViewNumber;
  CompilerID = &pgm_dbg_dta->Element[current_view].CompilerID;

  /* Give command to QteSubmitDebugCommand                           */
  QteSubmitDebugCommand(OutputBuffer, &OutputBufferLength,
                        &view_ID, InputBuffer, &InputBufferLength,
                        *CompilerID, &errorCode);

  if (errorCode.BytesAvailable != 0) {
    printf("Error = %.7s\n",errorCode.ExceptionID);
    return return_value;
  }

  /* Process results from QteSubmitDebugCommand                      */
  Results = (_TE_RESULT_BUFFER_T *) OutputBuffer;

  /* Loop through Results array                                      */
  for (i=0; i<Results->Header.EntryCount; i++) {
    switch (Results->Data[i].ResultKind)
    {
      case _TE_kStepR            :
        printf("Step set\n");
        return_value=1;            /* indicate step is to be done   */
        break;
      case _TE_kBreakR           :
        printf("Breakpoint set");
```

```
        break;
      case _TE_kBreakPositionR   :
        printf(" at line %d\n",
               Results->Data[i].V.BreakPosition.Line);
        break;
      case _TE_kExpressionTextR  :
        printf("%s",
               ((char *)Results) + Results->Data[i].V.
               ExpressionText.oExpressionText);
        break;
      case _TE_kExpressionValueR :
        printf(" = %s\n",
               ((char *)Results) + Results->Data[i].V.
               ExpressionValue.oExpressionValue);
        break;
      case _TE_kQualifyR         :
        printf("Qual set\n");
        break;
      case _TE_kClearBreakpointR :
        printf("Breakpoint cleared\n");
        break;
      case _TE_kClearPgmR :
        printf("All breakpoints cleared\n");
        break;
      default:                      /* ignore all other record types */
        break;
    }                               /* switch                        */
  }                                 /* loop through results array    */
  return return_value;
}

/* ProcessCommands:  Read input from user and process commands.      */
void ProcessCommands(void) {
  char InputBuffer[80];
  char *token;
  int i;
  int step=0;                       /* do an exit for step when 1    */

  if (pgm_dbg_dta == NULL) {        /* if no debug data              */
    printf("Debug session has ended.\n");
    exit(0);                        /* end the debugger              */
  }

  while(!step) {                    /* read until step or quit cmd   */
    ReadLine(InputBuffer,sizeof(InputBuffer));
    token = strtok(InputBuffer," ");

    if (token==NULL) continue;      /* ignore blank lines            */
    else if (strcmp(token,"quit") == 0) /* the quit command?         */
      return;                       /* exit debugger                 */
    else if (strcmp(token,"list") == 0) /* the list command?         */
      ProcessListCommand();         /* process command               */
    else if (strcmp(token,"switch") == 0) { /* switch command?       */
      token = strtok(NULL," ");     /* get view number token         */
      if (token == NULL)
        printf("'switch' must be followed by a view number.\n");
      else
        current_view = atoi(token);   /* switch current view         */
    }
    else if (strcmp(token,"help") == 0) {
      printf("The following are the allowed debugger commands:\n");
      printf("  list views - lists all views in the program\n");
      printf("  list text - lists the text of the current view\n");
      printf("  switch n - changes current view to view n\n\n");
      printf("  help - displays this help text\n");
      printf("  quit - ends the debug session\n");
      printf("Other commands are interpreted by the debug support.\n");
```

```
    }
    else {                          /* pass command to API           */
      /* Undo modifications that strtok did                          */
      InputBuffer[strlen(InputBuffer)] = ' ';

      step = ProcessDbgCommand(InputBuffer);
    }
  }
}

/* TearDownDebugger: End the debugger.                               */
void TearDownDebugger(void) {
  _TE_ERROR_CODE_T errorCode = {8}; /* errors will be ignored        */

  /* Call EndSourceDebug to get out of ILE debug mode               */
  QteEndSourceDebug(&errorCode);

  exit(0);                          /* destroy activation group      */
}

/* AddProgram: Add a program to debug mode.                          */
void AddProgram(void) {

  _TE_ERROR_CODE_T errorCode = {0};    /* Signal exceptions on error */
  _TE_NAME_T       Library;            /* Lib returned               */
  _TE_TIMESTAMP_T  TimeStamp;          /* TimeStamp returned         */
  int viewIndex;
  long int         iViewID;
  long int         iViewLines;
  long rtvModViewDataLength = 8;       /* size of receiver buffer    */
  char tempBuffer[8];                  /* Temp storage               */
  int i, tempModuleCount;

  /* Call QteRetrieveModuleViews to determine the number of bytes    */
  /* the receiver variable needs to be to hold all of the views for  */
  /* the program.                                                    */
  pgm_dbg_dta = (_TE_VEWL0100_T *)tempBuffer;
  QteRetrieveModuleViews((char *)pgm_dbg_dta, &rtvModViewDataLength,
                         "VEWL0100", &program_lib,
                         "*PGM      ", "*ALL      ", Library,
                         &errorCode);

  /* Get a buffer large enough to hold all view information          */
  rtvModViewDataLength = pgm_dbg_dta->BytesAvailable;
  pgm_dbg_dta = (_TE_VEWL0100_T *)malloc(rtvModViewDataLength);

  /* Call QteRetrieveModuleViews again, passing a big enough buffer. */
  QteRetrieveModuleViews((char *)pgm_dbg_dta, &rtvModViewDataLength,
                         "VEWL0100", &program_lib,
                         "*PGM      ", "*ALL      ", Library,
                         &errorCode);

  /* If number of elements is zero, program is not debuggable.       */
  if (pgm_dbg_dta->NumberElements == 0) {
    printf("Program %.10s in Library %.10s cannot be debugged.",
           program_lib.obj, program_lib.lib);
    TearDownDebugger();
  }

  /* Put the library returned by Retrieve Module Views in PgmLib     */
  memcpy(program_lib.lib, Library, sizeof(_TE_NAME_T));

  /* Register all views in the program                               */
  for (i=0; i < pgm_dbg_dta->NumberElements; i++) {
    QteRegisterDebugView(&iViewID, &iViewLines, Library, TimeStamp,
                         &program_lib, "*PGM      ",
                         pgm_dbg_dta->Element[i].ModuleName,
```

```
                        &pgm_dbg_dta->Element[i].ViewNumber,
                        &errorCode);

    /* overwrite unneeded ViewNumber with obtained view id          */
    pgm_dbg_dta->Element[i].ViewNumber = iViewID;
  }
}

/* Typedef for program list passed to this program at STRDBG time   */
typedef struct {
    _TE_OBJLIB_T PgmLib;            /* Name and Library of program   */
    _TE_NAME_T PgmType;             /* program type, *PGM or *SRVPGM */
} PgmList_t;

/* StartUpDebugger: Initialize the debugger.                        */
void StartUpDebugger(PgmList_t ProgramList[],
                     int ProgramListCount) {

  _TE_ERROR_CODE_T errorCode = {0}; /* exceptions are generated      */
  _TE_OBJLIB_T StopHandler = {"DEBUG     ", "*LIBL     "};
  int i;

  if (ProgramListCount!=1) {        /* is only 1 pgm passed on STRDBG*/
    printf("Exactly ONE program must be specified on STRDBG.\n");
    TearDownDebugger();             /* end debugger
 */
  }

  /* Copy program name to global variables                          */
  memcpy(&program_lib, &ProgramList->PgmLib, 20);

  /* Call StartSourceDebug: giving the name and library of the      */
  /* stop handler.  This will start ILE debug mode                  */
  QteStartSourceDebug(&StopHandler, &errorCode);

  AddProgram();                     /* add program to debug         */
}

/* HandleSession:  This function is called to handle the session    */
/* events STRDBG, DSPMODSRC and ENDDBG.                             */
void HandleSession(char reason[10],
                   PgmList_t ProgramList[],
                   int ProgramListCount) {

  if (memcmp(reason,"*START    ",10) == 0) /* reason is *START       */
    StartUpDebugger(ProgramList, ProgramListCount);
  else if ( memcmp(reason,"*STOP     ",10) == 0) /* reason is *STOP  */
    TearDownDebugger();
  else if ( memcmp(reason,"*DISPLAY  ",10) == 0) /* reason *DISPLAY  */
    ProcessCommands();
}

/* HandleStop: This function is called to handle stop events like   */
/* breakpoint, step, unmonitored exception, etc.                    */
void HandleStop(_TE_OBJLIB_T *ProgramLib,
                _TE_NAME_T ProgramType,
                _TE_NAME_T Module,
                char reason[10],
                long Statements[],
                int StatementsCount,
                char *message) {
  int i;
  _TE_MAPP0100_T Map_Return_Structure;
  long Column = 1;
  long MapLength = sizeof(Map_Return_Structure);
  _TE_ERROR_CODE_T errorCode = {64};
  long stmt_view;
```

```
  /* If current view is for a different module than the one that is  */
  /* stopped, change current view to first view in the stopped module*/
  if (memcmp(Module,
             pgm_dbg_dta->Element[current_view].ModuleName,
             sizeof(_TE_NAME_T)) != 0) { /* a different module?      */
    for (i=0; i<pgm_dbg_dta->NumberElements; i++) {
      if (memcmp(Module,
                 pgm_dbg_dta->Element[i].ModuleName,
                 sizeof(_TE_NAME_T)) == 0) { /* found module         */
        current_view = i;           /* change current view to module */
        printf("Current view changed to %d.\n",current_view);
        break;                      /* exit search loop              */
      }                             /* module found                  */
    }                               /* loop through views            */
  }                                 /* current view to be changed    */

  /* Get number of statement view for module stopped                 */
  for (i=0; i<pgm_dbg_dta->NumberElements; i++) {
    if ((memcmp(Module,
                pgm_dbg_dta->Element[i].ModuleName,
                sizeof(_TE_NAME_T)) == 0) &&
        (memcmp("*STATEMENT",
                  pgm_dbg_dta->Element[i].ViewType,
                  sizeof(_TE_NAME_T)) == 0))
      stmt_view = i;
  }


  /* Call QteMapViewPosition to map the stopped location (which      */
  /* is in terms of the *STATEMENT view) to the current view of      */
  /* the module                                                      */
  QteMapViewPosition((char *)&Map_Return_Structure, &MapLength,
                     &pgm_dbg_dta-> Element[stmt_view].ViewNumber,
                     &Statements[0], &Column,
                     &pgm_dbg_dta->Element[current_view].ViewNumber,
                     &errorCode);

  /* Tell the user about the program that stopped.                   */
  for (i=0;i<4;i++) {               /* See why program stopped       */
    if (reason[i] == '1') {
      switch(i) {
        case 0: printf("Unmonitored exception");
          break;
        case 1: printf("Breakpoint");
          break;
        case 2: printf("Step completed");
          break;
        case 3: printf("Breakpoint condition error");
          break;
      }
    }
  }
  printf(" in module %.10s at line %d.\n",
         Module,
         Map_Return_Structure.MapElem[0].LineNumber);

  ProcessCommands();                /* put user into debugger        */
}

/* main: Entry point for the debugger (session or stop handler)      */
main (int argc, char *argv[]) {
  if (argc == 4)                    /* called as source debug program*/
    HandleSession(argv[1], (PgmList_t *)argv[2], *(int
*)argv[3]);
  else if (argc == 8)             /* called as program stop handler */
    HandleStop((_TE_OBJLIB_T *)argv[1], argv[2],
```

```
argv[3], argv[4],
              (long *)argv[5], *(int *)argv[6],
argv[7]);
}
```

Example: Using process-related APIs

These ILE C programs perform process-related functions in a parent-child relationship.

See the QlgSpawn--Spawn Process (using NLS-enabled path name) API for an example of supplying parameters in any CCSID.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Parent program

This program acts as a parent to a child program (see "Child program" on page 456).

This program demonstrates the use of the spawn() function and the wait() and waitpid() functions in a parent/child relationship. The use of file descriptors, the creation of a new process group, arguments passed from parent to child, and environment variables are demonstrated. The parent program uses spawn() in three different ways.

Use the Create C Module (CRTCMOD) and the Create Program (CRTPGM) commands to create this program (see "Creating the parent and child programs" on page 459).

Call this program with no parameters (see "Calling the parent program" on page 459).

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  This program acts as a parent to a child program.      */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: putenv(), spawn(), wait(), waitpid()                   */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <errno.h>
#include <fcntl.h>
#include <spawn.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define MAP_NUM    5
#define ARGV_NUM   6
#define ENVP_NUM   2
#define CHILD_PGM  "QGPL/CHILD"

extern char **environ;

/* This is a parent program that will use spawn() in 3 different     */
/* ways for 3 different children.  A file is created that is         */
/* written to, both by the parent and the 3 children.  The end result*/
/* of the file will look something like the following:               */
/*     Parent writes    Child writes                                 */
```

```
/*     -------------    ---------------------------------------      */
/*           1          argv[0]  getppid()  getpgrp()  getpid()      */
/*           2          argv[0]  getppid()  getpgrp()  getpid()      */
/*           3          argv[0]  getppid()  getpgrp()  getpid()      */
/* The parent uses wait() or waitpid() to wait for a given child to  */
/* return and to retrieve the resulting status of the child when it  */
/* does return.                                                      */
int main(int argc, char *argv[])
{
    int    rc;                           /* API return code          */
    int    fd, fd_read;                  /* parent file descriptors  */
    char   fd_str[4];                   /* file descriptor string    */
    char   f_path_name[] = "A_File";             /* file pathname    */
    int    buf_int;                      /* write(), read() buffer   */
    char   buf_pgm_name[22];            /* read() program name buffer */
    char   spw_path[] = "/QSYS.LIB/QGPL.LIB/CHILD.PGM";
                                         /* spawn() *path            */
    int    spw_fd_count;                 /* spawn() fd_count         */
    int    spw_fd_map[MAP_NUM];
                                          /* spawn() fd_map[]        */
    struct inheritance spw_inherit;      /* spawn() *inherit         */
    char  *spw_argv[ARGV_NUM];
                                        /* spawn() *argv[]           */
    char  *spw_envp[ENVP_NUM];
                                        /* spawn() *envp[]           */
    int    seq_num;                      /* sequence number          */
    char   seq_num_str[4];              /* sequence number string    */
    pid_t  pid;                          /* parent pid              */
    char   pid_str[11];                 /* parent pid string        */
    pid_t  pgrp;                         /* parent process group    */
    char   pgrp_str[11];                /* parent process group string */
    pid_t  spw_child_pid[3];             /* 3 spawn() child pid     */
    pid_t  wt_child_pid[3];             /* 3 wait()/waitpid() child pid */
    int    wt_stat_loc[3];
                                       /* 3 wait()/waitpid() *stat_loc*/
    int    wt_pid_opt = 0;               /* waitpid() option        */
    char   env_return_val[16];
                                       /* environ var "return_val="  */

    memset(&spw_inherit,0x00,sizeof(spw_inherit));

    /* Get the pid and pgrp for the parent.                          */
    pid = getpid();
    pgrp = getpgrp();

    /* Format the pid and pgrp value into null-terminated strings.   */
    sprintf(pid_str, "%d", pid);
    sprintf(pgrp_str, "%d", pgrp);

    /* Create a file and maintain the file descriptor.               */
    fd = creat(f_path_name, S_IRWXU);
    if (fd == -1)
      {
        printf("FAILURE: creat() with errno = %d\n",errno);
        return -1;
      }

    /* Format the file descriptor into null-terminated string.       */
    sprintf(fd_str, "%d", fd);

    /* Set the spawn() child arguments that are common for each      */
    /* child.                                                        */
    /* NOTE: The child will always get argv[0] in the                */
    /* LIBRARY/PROGRAM notation, but the                             */
    /* spawn() argv[0] (spw_argv[0]                                  */
    /* in this case) must be non-NULL in order to allow additional   */
    /* arguments.  For this example, the character pointer spw_path  */
```

```
/* was chosen.                                                  */
/* NOTE: The parent pid and the parent process group are passed  */
/* to the child for demonstration purposes only.                */
spw_argv[0] = spw_path;
spw_argv[1] = pid_str;
spw_argv[2] = pgrp_str;
spw_argv[4] = fd_str;
spw_argv[5] = NULL;

/* Write a '1' out to the file.                                 */
buf_int = 1;
write(fd, &buf_int, sizeof(int));

/* The 1st spawn() will use simple inheritance for file         */
/* descriptors (fd_map[] value is NULL).                        */
spw_fd_count = 0;
spw_inherit.pgroup = 0;
seq_num = 1;
sprintf(seq_num_str, "%d", seq_num);
spw_argv[3] = seq_num_str;
spw_envp[0] = NULL;
spw_child_pid[0] = spawn(spw_path, spw_fd_count, NULL, &spw_inherit,
                         spw_argv, spw_envp);
if (spw_child_pid[0] == -1)
  {
    printf("FAILURE: spawn() #1 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* NOTE: The parent can continue processing while the child is   */
/* also processing.  In this example, though, the parent will    */
/* simply /* wait() until the child finishes processing.         */

/* Issue wait() in order to wait for the child to return.       */
wt_child_pid[0] = wait(&wt_stat_loc[0]);
if (wt_child_pid[0] == -1)
  {
    printf("FAILURE: wait() #1 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* Check to ensure the child's pid returned from spawn() is the  */
/* same as the child's pid returned from wait(), for which       */
/* status was returned.                                         */
if ( (spw_child_pid[0] != wt_child_pid[0]) )
    printf("FAILURE: spawn() #1 and wait() #1 pid not the same\n");

/* Check to ensure the child did not encounter an error         */
/* condition.                                                   */
if (WIFEXITED(wt_stat_loc[0]))
  {
    if (WEXITSTATUS(wt_stat_loc[0]) != 1)
        printf("FAILURE: wait() exit status = %d\n",
               WEXITSTATUS(wt_stat_loc[0]));
  }
else
    printf("FAILURE: unknown child #1 status\n");

/* Write a '2' out to the file.                                 */
buf_int = 2;
write(fd, &buf_int, sizeof(int));

/* The 2nd spawn() will use mapping for the file descriptor,    */
```

```
/* along with the inheritance option to create a new process     */
/* group for the child.                                          */
spw_fd_count = 1;
spw_fd_map[0] = fd;
spw_inherit.pgroup = SPAWN_NEWPGROUP;
seq_num = 2;
sprintf(seq_num_str, "%d", seq_num);
spw_argv[3] = seq_num_str;
spw_envp[0] = NULL;
spw_child_pid[1] = spawn(spw_path, spw_fd_count, spw_fd_map,
                         &spw_inherit, spw_argv, spw_envp);
if (spw_child_pid[1] == -1)
  {
    printf("FAILURE: spawn() #2 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* NOTE: The parent can continue processing while the child is   */
/* also processing.  In this example, though, the parent will    */
/* simply waitpid() until the child finishes processing.         */

/* Issue waitpid() in order to wait for the child to return.     */
wt_child_pid[1] = waitpid(spw_child_pid[1], &wt_stat_loc[1],
                          wt_pid_opt);
if (wt_child_pid[1] == -1)
  {
    printf("FAILURE: waitpid() #2 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* Check to ensure the child's pid returned from spawn() is the  */
/* same as the child's pid returned from waitpid(), for which    */
/* status was returned.                                          */
if ( (spw_child_pid[1] != wt_child_pid[1]) )
    printf("FAILURE: spawn() #2 and waitpid() #2 pid not same\n");

/* Check to ensure the child did not encounter an error          */
/* condition.                                                    */
if (WIFEXITED(wt_stat_loc[1]))
  {
    if (WEXITSTATUS(wt_stat_loc[1]) != 2)
        printf("FAILURE: waitpid() exit status = %d\n",
               WEXITSTATUS(wt_stat_loc[1]));
  }
else
    printf("FAILURE: unknown child #2 status\n");

/* Write a '3' out to the file.                                  */
buf_int = 3;
write(fd, &buf_int, sizeof(int));

/* The 3rd spawn() will use mapping for the file descriptors     */
/* with some file descriptors designated as being closed         */
/* (SPAWN_FDCLOSED) and the same parent file descriptor mapped   */
/* to more than one child file descriptor.  In addition, an      */
/* environment variable will be set and used by the child.       */
spw_fd_count = 5;
spw_fd_map[0] = SPAWN_FDCLOSED;
spw_fd_map[1] = SPAWN_FDCLOSED;
spw_fd_map[2] = fd;
spw_fd_map[3] = SPAWN_FDCLOSED;
spw_fd_map[4] = fd;
spw_inherit.pgroup = 0;
```

```
seq_num = 3;
sprintf(seq_num_str, "%d", seq_num);
spw_argv[3] = seq_num_str;
strcpy(env_return_val,"return_val=3");
rc = putenv(env_return_val);
if (rc < 0)
  {
    printf("FAILURE: putenv() with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }
spw_child_pid[2] = spawn(spw_path, spw_fd_count, spw_fd_map,
                         &spw_inherit, spw_argv, environ);
if (spw_child_pid[2] == -1)
  {
    printf("FAILURE: spawn() #3 with errno = %d\n",errno);
    close(fd);
    unlink(f_path_name);
    return -1;
  }

/* The parent no longer needs to use the file descriptor, so it  */
/* can close it, now that it has issued spawn().                 */
rc = close(fd);
if (rc != 0)
    printf("FAILURE: close(fd) with errno = %d\n",errno);

/* NOTE: The parent can continue processing while the child is   */
/* also processing.  In this example, though, the parent will    */
/* simply wait() until the child finishes processing.            */

/* Issue wait() in order to wait for the child to return.        */
wt_child_pid[2] = wait(&wt_stat_loc[2]);
if (wt_child_pid[2] == -1)
  {
    printf("FAILURE: wait() #3 with errno = %d\n",errno);
    unlink(f_path_name);
    return -1;
  }

/* Check to ensure the child's pid returned from spawn() is the  */
/* same as the child's pid returned from wait(), for which       */
/* status was returned.                                          */
if ( (spw_child_pid[2] != wt_child_pid[2]) )
    printf("FAILURE: spawn() #3 and wait() #3 pid not the same\n");

/* Check to ensure the child did not encounter an error          */
/* condition.                                                    */
if (WIFEXITED(wt_stat_loc[2]))
  {
    if (WEXITSTATUS(wt_stat_loc[2]) != 3)
        printf("FAILURE: wait() exit status = %d\n",
               WEXITSTATUS(wt_stat_loc[2]));
  }
else
    printf("FAILURE: unknown child #3 status\n");

/* Open the file for read to verify what the child wrote.        */
fd_read = open(f_path_name, O_RDONLY);
if (fd_read == -1)
  {
    printf("FAILURE: open() for read with errno = %d\n",errno);
    unlink(f_path_name);
    return -1;
  }
```

```
/* Verify what child #1 wrote.                                        */
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != 1) )
    printf("FAILURE: read() #1\n");
memset(buf_pgm_name,0x00,sizeof(buf_pgm_name));
rc = read(fd_read, buf_pgm_name, strlen(CHILD_PGM));
if ( (rc != strlen(CHILD_PGM)) ||
     (strcmp(buf_pgm_name,CHILD_PGM) != 0) )
    printf("FAILURE: read() child #1 argv[0]\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != pid) )
    printf("FAILURE: read() child #1 getppid()\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != pgrp) )
    printf("FAILURE: read() child #1 getpgrp()\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != spw_child_pid[0]) ||
     (buf_int != wt_child_pid[0]) )
    printf("FAILURE: read() child #1 getpid()\n");

/* Verify what child #2 wrote.                                        */
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != 2) )
    printf("FAILURE: read() #2\n");
memset(buf_pgm_name,0x00,sizeof(buf_pgm_name));
rc = read(fd_read, buf_pgm_name, strlen(CHILD_PGM));
if ( (rc != strlen(CHILD_PGM)) ||
     (strcmp(buf_pgm_name,CHILD_PGM) != 0) )
    printf("FAILURE: read() child #2 argv[0]\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != pid) )
    printf("FAILURE: read() child #2 getppid()\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int == pgrp) )
    printf("FAILURE: read() child #2 getpgrp()\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != spw_child_pid[1]) ||
     (buf_int != wt_child_pid[1]) )
    printf("FAILURE: read() child #2 getpid()\n");

/* Verify what child #3 wrote.                                        */
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != 3) )
    printf("FAILURE: read() #3\n");
memset(buf_pgm_name,0x00,sizeof(buf_pgm_name));
rc = read(fd_read, buf_pgm_name, strlen(CHILD_PGM));
if ( (rc != strlen(CHILD_PGM)) ||
     (strcmp(buf_pgm_name,CHILD_PGM) != 0) )
    printf("FAILURE: read() child #3 argv[0]\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != pid) )
    printf("FAILURE: read() child #3 getppid()\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != pgrp) )
    printf("FAILURE: read() child #3 getpgrp()\n");
rc = read(fd_read, &buf_int, sizeof(int));
if ( (rc != sizeof(int)) || (buf_int != spw_child_pid[2]) ||
     (buf_int != wt_child_pid[2]) )
    printf("FAILURE: read() child #3 getpid()\n");

/* Attempt one more read() to ensure there is no more data.           */
rc = read(fd_read, &buf_int, sizeof(int));
if (rc != 0)
    printf("FAILURE: read() past end of data\n");

/* The parent no longer needs to use the read() file descriptor,      */
/* so it can close it.                                                */
```

```
    rc = close(fd_read);
    if (rc != 0)
        printf("FAILURE: close(fd_read) with errno = %d\n",errno);

    /* Attempt one more wait() to ensure there are no more children. */
    wt_child_pid[0] = wait(&wt_stat_loc[0]);
    if ( (wt_child_pid[0] != -1) || (errno != ECHILD) )
        printf("FAILURE: ECHILD wait()\n");

    /* Clean up by performing unlink().                              */
    rc = unlink(f_path_name);
    if (rc != 0)
      {
        printf("FAILURE: unlink() with errno = %d\n",errno);
        return -1;
      }
    printf("completed successfully\n");
    return 0;
}
```

Child program

This program acts as a child to a parent program (see "Parent program" on page 450). This program demonstrates how a child program uses characteristics expressed through the use of spawn() in the parent program. The use of file descriptors, the creation of a new process group, arguments passed from the parent, and environment variables are demonstrated. The child program handles three distinct calls through the use of one of its arguments.

Use the CRTCMOD and CRTPGM commands to create this program (see "Creating the parent and child programs" on page 459).

This program is called by the spawn() function from the parent program. The program name must be CHILD and must be created into library QGPL, as indicated by the parent program. This program is not to be called directly.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/* FUNCTION:  This program acts as a child to a parent program.      */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/* APIs USED: getenv(), getpid(), getppid(), getpgrp()               */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/

#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

/* This is a child program that gets control from a parent program   */
/* that issues spawn().  This particular child program expects the   */
/* following 5 arguments (all are null-terminated strings):          */
/*     argv[0] - child program name                                  */
/*     argv[1] - parent pid (for demonstration only)                 */
/*     argv[2] - parent process group (for demonstration only)       */
/*     argv[3] - sequence number                                     */
/*     argv[4] - parent file descriptor                              */
/* If the child program encounters an error, it returns with a value */
/* greater than 50.  If the parent uses wait() or waitpid(), this    */
/* return value can be interrogated using the WIFEXITED and          */
/* WEXITSTATUS macros on the resulting wait() or waitpid()           */
/* *stat_loc field.                                                  */
```

```
int main(int argc, char *argv[])
{
    pid_t  p_pid;                   /* parent pid argv[1]           */
    pid_t  p_pgrp;                  /* parent process group argv[2] */
    int    seq_num;                 /* parent sequence num argv[3]  */
    int    fd;                      /* parent file desc argv[4]     */
    int    rc;                         /* API return code           */
    pid_t  pid;                        /* getpid() - child pid      */
    pid_t  ppid;                       /* getppid() - parent pid    */
    pid_t  pgrp;                       /* getpgrp() - process group */
    char  *env_return_val;          /* environ var for "return_val" */

    /* Get the pid, ppid, and pgrp for the child.                    */
    pid = getpid();
    ppid = getppid();
    pgrp = getpgrp();

    /* Verify 5 parameters were passed to the child.                 */
    if (argc != 5)
        return 60;

    /* Since the parameters passed to the child using spawn() are    */
    /* pointers to strings, convert the parent pid, parent process   */
    /* group, sequence number, and the file descriptor from strings  */
    /* to integers.                                                  */
    p_pid = atoi(argv[1]);
    p_pgrp = atoi(argv[2]);
    seq_num = atoi(argv[3]);
    fd = atoi(argv[4]);

    /* Verify the getpid() value of the parent is the same as the    */
    /* getppid() value of the child.                                 */
    if (p_pid != ppid)
        return 61;

    /* If the sequence number is 1, simple inheritance was used in   */
    /* this case.  First, verify the getpgrp() value of the parent   */
    /* is the same as the getpgrp() value of the child.  Next, the   */
    /* child will use the file descriptor passed in to write the     */
    /* child's values for argv[0], getppid(), getpgrp(),            */
    /* and getpid().  Finally, the child returns, which will satisfy */
    /* the parent's wait() or waitpid().                             */
    if (seq_num == 1)
      {
        if (p_pgrp != pgrp)
            return 70;
        rc = write(fd, argv[0], strlen(argv[0]));
        if (rc != strlen(argv[0]))
            return 71;
        rc = write(fd, &ppid, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 72;
        rc = write(fd, &pgrp, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 73;
        rc = write(fd, &pid, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 74;
        return seq_num;
      }

    /* If the sequence number is 2, file descriptor mapping was used */
    /* in this case.  In addition, an inheritance option was used to */
    /* indicate this child will create a new process group.  First,  */
    /* verify the getpgrp() value of the parent is different than    */
    /* the getpgrp() value of the child.  Next, the child will use   */
    /* a literal value of '0' as the file descriptor (instead of the */
```

```
    /* parent's file descriptor passed in) since a known mapping was */
    /* performed by the parent.  This literal is used to write the   */
    /* child's values for argv[0], getppid(), getpgrp(),             */
    /* and getpid().  Finally, the child returns, which will satisfy */
    /* the parent's wait() or waitpid().                             */
    else if (seq_num == 2)
      {
        if (p_pgrp == pgrp)
            return 80;
        rc = write(0, argv[0], strlen(argv[0]));
        if (rc != strlen(argv[0]))
            return 81;
        rc = write(0, &ppid, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 82;
        rc = write(0, &pgrp, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 83;
        rc = write(0, &pid, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 84;
        return seq_num;
      }

    /* If the sequence number is 3, file descriptor mapping was used */
    /* in this case.  In addition, an environment variable by the    */
    /* name of "return_val" was set with the desired return value.   */
    /* First, verify the getpgrp() value of the parent is the same   */
    /* as the getpgrp() value of the child.  Next, the child will    */
    /* use literal values of '2' and '4' as the file descriptor      */
    /* (instead of the parent's file descriptor passed in) since a   */
    /* known mapping was performed by the parent.  These literals    */
    /* are used to write the child's values for argv[0], getppid(),  */
    /* getpgrp(), and getpid().  Finally, getenv() is performed to   */
    /* retrieve the desired value to use on return, which will       */
    /* satisfy the parent's wait() or waitpid().                     */
    else if (seq_num == 3)
      {
        if (p_pgrp != pgrp)
            return 90;
        rc = write(4, argv[0], strlen(argv[0]));
        if (rc != strlen(argv[0]))
            return 91;
        rc = write(2, &ppid, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 92;
        rc = write(4, &pgrp, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 93;
        rc = write(2, &pid, sizeof(pid_t));
        if (rc != sizeof(pid_t))
            return 94;
        env_return_val = getenv("return_val");
        return (atoi(env_return_val));
      }

    /* If the sequence number is an unexpected value, return         */
    /* indicating an error.                                          */
    else
        return 99;
}
```

Creating the parent and child programs

The following examples show how to create the example programs ("Parent program" on page 450 and "Child program" on page 456). These examples assume that the source for the parent program is member PARENT in the file QGPL/QCSRC and the source for the child program is member CHILD in the file QGPL/QCSRC.

Creating the parent module:

```
CRTCMOD MODULE(QGPL/PARENT)
        SRCFILE(QGPL/QCSRC)
        SRCMBR(PARENT)
        TEXT('Example Parent')
```

Creating the child module:

```
CRTCMOD MODULE(QGPL/CHILD)
        SRCFILE(QGPL/QCSRC)
        SRCMBR(CHILD)
        TEXT('Example Child')
```

Creating the parent program:

```
CRTPGM PGM(QGPL/PARENT)
```

Creating the child program:

```
CRTPGM PGM(QGPL/CHILD)
```

Calling the parent program

The following example shows how to start the example programs:

```
CALL PGM(QGPL/PARENT)
```

Related reference:

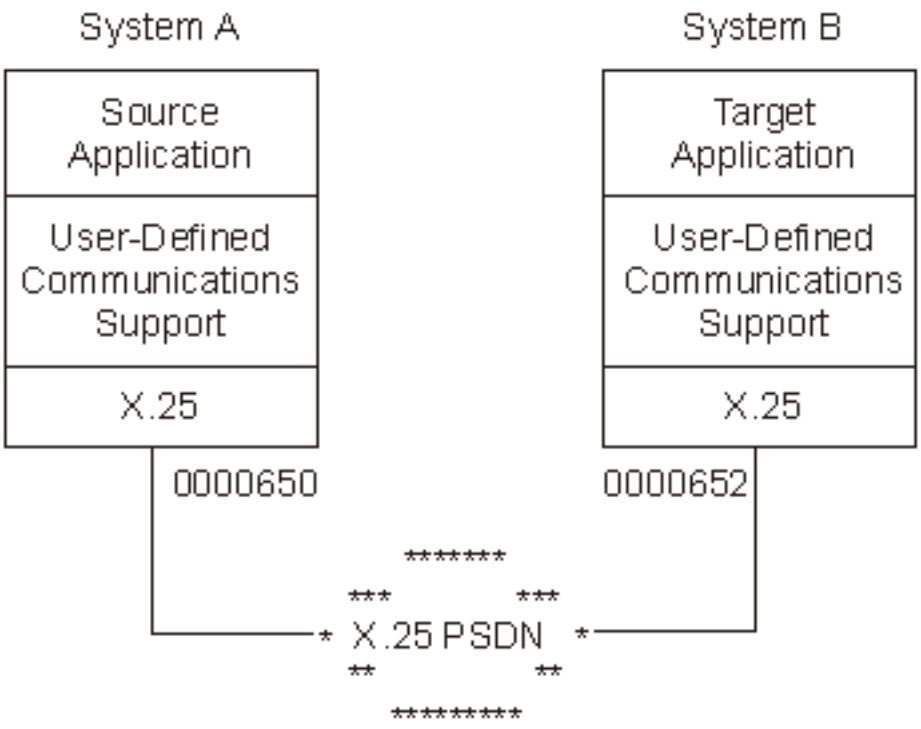
QlgSpawn()--Spawn Process (using NLS-enabled path name) API

Example: Using the user-defined communications programs for file transfer

This example shows how X.25-oriented applications use the user-defined communications support to connect to remote systems. Two user-defined application programs, written in ILE C, are used to illustrate a simple file transfer between 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. The includes *header*, *hexconv*, and *typedefs* are not in QSYSINC. These includes are only documented in the examples.

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 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 (SRC) and target (TGT) 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. 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.

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.

ILE C 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 in "Source application program listing references" on page 472 and "Target application program listing references" on page 484.

The target application compiler listing can be found in Target application on System B listing.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

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 using 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

To create the program using ILE C, use the Create Bound C (CRTBNDC) command.

```
 Program name  . . . . . :
(SOURCE)
   Library name  . . . . . :     UDCS_APPLS
 Source file . . . . . . . :   QCSRC
   Library name  . . . . . :     UDCS_APPLS
 Source member name  . . . . . :   SOURCE
 Text Description  . . . . . :   Source Application Example
 Output  . . . . . . . . . . . :   *NONE
 Compiler options  . . . . . :   *SOURCE  *NOXREF  *SHOWUSR
                               :   *SHOWSYS  *NOSHOWSKP  *NOEXPMAC
                               :   *NOAGR *NOPPONLY  *NODEBUG
                               :   *GEN *NOSECLVL  *PRINT  *LOGMSG
                               :   *USRINCPATH
 Checkout Options  . . . . . :   *NOAGR
 Optimization  . . . . . :   *NONE
 Inline Options:
   Inliner . . . . . :   *OFF
   Mode  . . . . . :   *NOAUTO
   Threshold . . . . . :   250
   Limit . . . . . :   2000
 Debugging View  . . . . . :   *NONE
 Define Names  . . . . . :   *NONE
 Language Level  . . . . . :   *SOURCE
 Source Margins:
   Left margin . . . . . :   1
   Right margin  . . . . . :   32754
```

```
 Sequence columns:
   Left column . . . . . :   *NONE
   Right column  . . . . . :
 Message flagging level  . . . . :   0
 Compiler messages:
   Message limit . . . . . :   *NOMAX
   Message limit severity  . . . :   30
 Replace Program Object  . . . . :   *YES
 User Profile  . . . . . :   *USER
 Authority . . . . . :   *LIBCRTAUT
 Target Release  . . . . . :   *CURRENT
 System includes . . . . . :   *YES

/*****************************************************************/
/**  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.                       **/
/**                                                             **/
/**                                                             **/
/** 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.                                                     **/
/**                                                             **/
/** Language:  ILE C                                            **/
/**                                                             **/
```

```
/** APIs used: QOLELINK, QUSPTRUS, QOLRECV, QOLSEND, QOLDLINK,    **/
/**            QOLTIMER, QRCVDTAQ                                  **/
/**                                                                **/
/********************************************************************/
/********************************************************************/
/********************************************************************/
#include "header"
#include "typedef"
#include "hexconv"
```
(1)

```
/************   Typedef  Declarations      ******************/
```
(2)

```
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);

void _GetExcData(_INTRPT_Hndlr_Parms_T *parms);
/************************************************************/
/***************   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;
```
(3)

```
  /***---  Open the file to send to remote side        ----**/
  if ((fptr = fopen("UDCS_APPLS/INFILE(INFILE)", "r")) == NULL)
     {
     printf("Unable to open source input file in UDCS_APPLS LIB.\n");
     printf("The Program was terminated.\n\n");
```

```
   return;
   }
/***---  Open the display file as our input screen.  ----**/
if ((screen = fopen("ERRORSPEC", "ab+ type=record")) == NULL)
   {
   printf("Unable to open display file.\n");
   printf("The Program was terminated.\n\n");
   return;
   }
/** set the exception handler  **/

signal(SIGALL,SIG_DFL);
/**  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);
```

(4)

```
/************************************************/
/************  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;
   }
```

(5)

```
/*-------- Set a timer for Enable Link  ---------**/
expctid = 0xF0F0;
settimer(&expctid, "Enable", &dataq, &qname, commhandle);
if (expctid != 0xF0F0)
   {
   disablelink (&disable, commhandle, &qname);
   return;
   }
```

(6)

```
/*********************************************************************/
/**************   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;
   }
```

(7)

```
/*********************************************************************/
/***********   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;
   }
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");
```

(8)

```
/*****************************************************************/
/***************   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  **************/
/*****************************************************************/
```

(9)

```
/****   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;
   }
```

(10)

```
/*****************************************************************/
/***********   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("******   SOURCE completed successfully   ******\n\n");
}    /* End Main */
/*****************************************************************/
/************** Start Subroutine Section    **********************/
/*****************************************************************/
/*************************************************************/
/***************     Send a Packet of Data  ******************/
```
(11)

```
void senddata (sendparms *send,
               char *buffer,
               desc *descriptor,
               char *commhandle,
               char *line,
               int linesiz)
{
  descriptor->length = linesiz;
  descriptor->more = 0;
  descriptor->qualified = 0;
  descriptor->interrupt = 0;
  descriptor->dbit = 0;
  strncpy (buffer, line, linesiz);
  QOLSEND (&(send->retcode), &(send->reason),
&(send->errorspecific),\
           &(send->newpcep), &(send->ucep), &(send->pcep), \
           commhandle, &(send->operation), &(send->numdtaelmnts));
}  /* End senddata Subroutine  */
/*********************************************************/
/************  Routine to fill X.25 Format I   ***********/
void sndformat1 (sendparms *send,
                 char *buffer,
                 char *rmtdte,
                 char *commhandle,
                 qlindparms *qlind)
{
  format1 *output = (format1 *) buffer;
  register int counter;
  register querydata *qd;
  qd = (querydata *)&(qlind->userbuffer);
  output->type = 2;                /*  SVC used  */
```

```
  output->logchanid = 0x0;
  output->sendpacksize = qd->x25data.defsend;
  output->sendwindsize = qd->x25data.windowsend;
  output->recvpacksize = qd->x25data.defrecv;
  output->recvwindsize = qd->x25data.windowrecv;
  output->dtelength = strlen(rmtdte);
  byte(output->dte, 16, rmtdte, strlen(rmtdte));
  output->dbit = 0;
  output->cug = 0;
  output->cugid = 0;
  output->reverse = 0;
  output->fast = 0;
  output->faclength = 0;
  byte(output->facilities, 109, "", 0);
  output->calllength = 1;
  byte(output->callud, 128, "21", 2);   /* Contains Remote PID */
  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  */
```

(12)

```
/*****************************************/
/********    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  */
/************************************************************/
/**    Routine to display the ErrorSpecific output       ******/
void printespec(espec *errorspecific)
{
  especout outparms;

  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  */
```

(13)

```
/*************   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;
  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);
  /*** 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  */
```
(14)

```
/*********************************************************/
/**  x25lind:  Retrieve X.25 line description information  **/
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  */
/*************************************************/
```

Source application program listing references

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.
(2)	Function prototypes of the internal functions used in this program.
(3)	Call the C library routines fopen() and signal() to open the source file and set up a signal handler to process IBM i exceptions, respectively. An example of an exception would be accessing a data area with a NULL pointer. If an exception situation is encountered, SIG_DFL, the default handler, will be called in order for the program to end.
(4)	Call the QOLQLIND API to retrieve local configuration information from the line description about what 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.
(5)	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.

(6)	Call the QOLSEND API with a X'B000' operation to establish a connection to the target application program.
(7)	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.
(8)	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.
(9)	Call the QOLSEND API with a X'B100' operation to clear the peer connection.
(10)	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.
(11)	The following C functions illustrate the various user-defined communications support APIs.
(12)	This procedure illustrates a call to the QOLDLINK API. The vary option is set to vary off the associated *USRDFN network device.
(13)	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.
(14)	The x25lind() illustrates calling the QOLQLIND API.

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 using 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

To create the program using ILE C, use the Create Bound C (CRTBNDC) command.

Explanations of the reference numbers in the listing can be found in “Target application program listing references” on page 484.

```
 Program name  . . . . . :
(TARGET)
   Library name  . . . . . :     UDCS_APPLS
 Source file . . . . . . . :   QCSRC
   Library name  . . . . . :     UDCS_APPLS
 Source member name  . . . . :   TARGET
 Text Description  . . . . . :  Target Application Example
 Output  . . . . . . . . . . :   *NONE
 Compiler options  . . . . . :   *SOURCE  *NOXREF *NOSHOWUSR
                             :   *NOSHOWSYS *NOSHOWSKP *NOEXPMAC
                             :   *NOAGR *NOPPONLY *NODEBUG
                             :   *GEN  *NOSECLVL  *PRINT  *LOGMSG
                             :   *USRINCPATH
 Checkout Options  . . . . . :   *NOAGR
 Optimization  . . . . . . . :   *NONE
 Inline Options:
   Inliner . . . . . . . . . :   *OFF
   Mode  . . . . . . . . . . :   *NOAUTO
   Threshold . . . . . . . . :   250
   Limit . . . . . . . . . . :   2000
 Debugging View  . . . . . . :   *NONE
 Define Names  . . . . . . . :   *NONE
 Language Level  . . . . . . :   *SOURCE
 Source Margins:
   Left margin . . . . . . . :   1
   Right margin  . . . . . . :   32754
 Sequence columns:
   Left column . . . . . . . :   *NONE
   Right column  . . . . . . :
 Message flagging level  . . . :   0
 Compiler messages:
   Message limit . . . . . . :   *NOMAX
   Message limit severity  . . :   30
 Replace Program Object  . . . :   *YES
 User Profile  . . . . . . . :   *USER
 Authority . . . . . . . . . :   *LIBCRTAUT
 Target Release  . . . . . . :   *CURRENT
 System includes . . . . . . :   *YES

/*****************************************************************/
/**                                                             **/
/**  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.         **/
/**                                                               **/
/**      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.                                                       **/
/**                                                               **/
/** Language:  ILE C                                              **/
/**                                                               **/
/** APIs used: QOLELINK, QUSPTRUS, QOLRECV, QOLSEND, QOLDLINK,    **/
/**            QRCVDTAQ, QOLTIMER                                 **/
/**                                                               **/
/*******************************************************************/
/*******************************************************************/
/*******************************************************************/
/*******************************************************************/
#include "header"
#include "typedef"
#include "hexconv"
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);

void _GetExcData(_INTRPT_Hndlr_Parms_T *parms);
/*********************************************************/
/***************   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;
  /******  Annnddddd....  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(SIGALL,SIG_DFL);
  /**  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;
     }
(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  ***/
     }
```

(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;
   }
```

(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  ***/
   }
```

(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 */
/*****************************************************************/
/************** 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  */
/*****************************************************************/
/**************   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)
{
  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;

  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);
  /*  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 QOLTIMER   */
  QOLTIMER (&(timer.retcode), &(timer.reason), timer.handleout,\
           timer.handlein, (char *)qname, &(timer.operation),\
           &(timer.interval), &(timer.establishcount),\
           &(timer.keylength), timer.keyvalue, timer.userdata);
  if ((timer.retcode != 0) || (timer.reason != 0))
```

```
     {
     printf("%s timer failed while being set.\n", process);
     printf("Return code = %d\n", timer.retcode);
     printf("Reason code = %d\n\n", timer.reason);
     }
  /**-------  Dequeue an entry  --------**/
  strncpy(key, "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  */
     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  */
/************************************************************/
/**  x25lind:  Read a record into buf and return length   **/
void x25lind (qlindparms *qlind, char *linename)
{
register int counter;
  for(counter=0;counter<256;counter++)
     qlind->userbuffer[counter]=0;
  qlind->format = 0x01;
  QOLQLIND (&(qlind->retcode), &(qlind->reason), &(qlind->nbytes),\
            qlind->userbuffer, linename, &(qlind->format));
}  /*  End x25lind Subroutine  */
/************************************************************/
/**  putdata:  Read a record into buf and return length   **/
void putdata (char *buf,
              int dtalen,
              FILE *fptr)
{
int i;
   for (i = 0; i < dtalen; i++)
      fwrite(buf + i, 1, 1, fptr);
}  /*  End putdata Subroutine  */
```

Target application program listing references

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 IBM i 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.

Includes for source and target programs

The following three includes are used by both the preceding source and target programs. They are not in an IBM i library.

```
/*********************************************************************/
/*********************************************************************/
/* Include Name: Header                                              */
/*                                                                   */
/*                                                                   */
/* Function:                                                         */
/*   Type define and declare the structures used to interface        */
/*   to the user-defined communications APIs.  These structures      */
/*   are used by both the source and target application.             */
/*                                                                   */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
```

```
/*                                                                  */
/* APIs USED:  QOLDLINK, QOLELINK, QOLSEND, QOLRECV, QOLSETF,       */
/*             QOLTIMER, QUSPTRUS, QRCVDTAQ, QCLRDTAQ, QOLQLIND     */
/*                                                                  */
/********************************************************************/
/********************************************************************/

FILE *screen;
FILE *rptr;
FILE *fptr;

#include <qoldlink.h>
#include <qolelink.h>
#include <qolsend.h>
#include <qolrecv.h>
#include <qolsetf.h>
#include <qoltimer.h>
#include <qusptrus.h>
#include <qrcvdtaq.h>
#include <qclrdtaq.h>
#include <qolqlind.h>

/************   Typedef  Declarations      *******************/

typedef struct usrspace
    {
      char name[10];
      char library[10];
    } usrspace;

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;

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 parameters  */
  {
   int retcode,             /*  Output  */
       reason,              /*  Output  */
       nbytes;              /*  Output  */
   char userbuffer[256];
```

```
   char format;
  } qlindparms;

typedef _Packed union content       /*  Queue support parameters  */
  {
    _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;

typedef _Packed struct qentry      /*  Queue  parameters  */
  {
    char type[10];
    unsigned short msgid;
    content message;
    char key[256];
  } qentry;
```

The following typedef include has new type declarations used by both source and target programs.

```
/*********************************************************************/
/*********************************************************************/
/* Include Name: Typedef                                             */
/*                                                                   */
/* Function:                                                         */
/*   Define the buffer spaces used to pass the data to the           */
/*   APIs.                                                           */
/*                                                                   */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
/*********************************************************************/
/*********************************************************************/
/*These definitions and C library #include files are either global, or
  are used by multiple modules in the Open FM API driver.*/

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <xxcvt.h>
#include <string.h>
#include <ctype.h>
#include <recio.h>

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;

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;

typedef struct querydata
  {
   char header[12];    /* line header info */
   x25info x25data;        /* preliminary data */
  } querydata;
/*********************************************************************/
/*********************************************************************/
/* Include Name: Hexconv                                             */
/*                                                                   */
/* Function:                                                         */
/*   This include brings in procedures to convert hexadecimal        */
/*   to integer values and vice versa.                               */
/*                                                                   */
/*                                                                   */
/* LANGUAGE:  ILE C                                                  */
/*                                                                   */
```

```
/*********************************************************************/
/*********************************************************************/
#include <stdio.h>

unsigned int hextoint(char *);

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);
}
```

Related reference:

Set Filter (QOLSETF) API

Example: Working with stream files

This ILE C program writes data from an existing stream file to a new database file.

The program uses the following hierarchical file system (HFS) APIs:

- Create Directory (QHFCRTDR)
- Open Stream File (QHFOPNSF)
- Read from Stream File (QHFRDSF)
- Close Stream File (QHFCLOSF)

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/* Program Name:  HFSCOPY                                            */
/* Language    :  ILE C                                              */
/* 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                      */
/* APIs Used   :  QHFCRTDR, QHFOPNSF, QHFRDSF, QHFCLOSF              */
/*********************************************************************/

/*********************************************************************/
/* Include files                                                     */
/*********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <qhfopnsf.h>
#include <qhfrdsf.h>
#include <qhfclosf.h>
#include <qhfcrtdr.h>
#include <qusec.h>
```

```
/*********************************************************************/
/* Structure and variable definitions                                */
/*********************************************************************/

#define ON   1
#define OFF  0
typedef struct error_code_struct {
                Qus_EC_t EC;
                char exception_data[256];
               }error_code_struct;
error_code_struct error_code;
char file_handle[16];
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->EC.Exception_Id;
   printf("Bytes Provided -> %d\n",theErrCode->EC.Bytes_Provided);
   printf("Bytes Available -> %d\n",theErrCode->EC.Bytes_Available);
   printf("Exception ID -> ");
   for (i=0;i<7 ;i++,tempptr++ )
     {
       putchar(*tempptr);
     }
   putchar('\n');
}


/*********************************************************************/
/* Start of code                                                     */
/*********************************************************************/
main()
{

 error_code.EC.Bytes_Provided = 116;
/*********************************************************************/
/* Create the directory                                              */
/*********************************************************************/
 strcpy(path_name,"/QDLS/HFSFLR");
 path_length = strlen(path_name);

 QHFCRTDR(path_name,path_length,&attrib_info,attrib_length,&error_code);
 if ( error_code.EC.Bytes_Available != 0 )
   {
    if (!memcmp(error_code.EC.Exception_Id,"CPF1F04",7))
       printf("Directory HFSFLR already created.\n");
    else
       {
        printErrCode(&error_code);
        exit(1);
       }
```

```
   }

/*******************************************************************/
/* Open the stream file                                            */
/*******************************************************************/
 strcpy(open_info,"210 120   ");    /* Create or replace the file   */
 strcpy(path_name,"/QDLS/HFSFLR/SAMPLE.HFS");
 path_length = strlen(path_name);
 printf("OPEN STREAM FILE:\n  ");
 QHFOPNSF(&file_handle,
          path_name,
          path_length,
          open_info,
          &attrib_info,
          attrib_length,
          &action,
          &error_code);
 if (error_code.EC.Bytes_Available != 0)
   {
    printErrCode(&error_code);
    exit(1);
   }

/*******************************************************************/
/* Open a database file                                            */
/*******************************************************************/
 system("CRTLIB LIB(HFSLIB)");
 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.EC.Bytes_Available != 0)
       {
         cmpgood = strncmp("CPF1F33",error_code.EC.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.EC.Bytes_Available != 0)
   printErrCode(&error_code);

/*********************************************************************/
/* Close the database file                                           */
/*********************************************************************/
 fclose(FP);
}
```

To create the program using ILE C, specify the following:

```
CRTBNDC PGM(QGPL/HFSCOPY) SRCFILE(QGPL/QCSRC)
```

Example: Creating a program temporary fix exit program

This example exit program, written in CL, covers these possible changes in the logical state of a program temporary fix (PTF): loaded to temporarily applied and temporarily applied to temporarily 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 or the not-applied to remove-permanent cases. It is assumed that all action was taken on the moves from loaded to apply-temporary and from apply-temporary to not-applied. If additional actions are necessary, code could be added to handle those transitions as well.

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.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/****  START OF SPECIFICATIONS  ***************************************/
/*                                                                     */
/* LANGUAGE:  CL                                                       */
/*                                                                     */
/* APIs USED: None                                                     */
/*                                                                     */
/* 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            */
/*                               4 - PRE-REMOVE TEMPORARILY        */
/*                               5 - PRE-APPLY TEMPORARILY         */
/*                               6 - PRE-APPLY PERMANENTLY         */
/*                               7 - PRE-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 = '5') THEN(DO)    /* If action will be temporarily */
                                   /* apply then                    */
        /*?---- PRE-TEMP 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 = '4') THEN(DO)    /* If action will be temporarily */
                                    /*  remove then                  */
        /*?---- PRE-TEMP REMOVE - ADD YOUR STATEMENTS HERE -----    */
     ENDDO
   ENDDO                            /* End of remove the PTF         */
    /*-----------------------------------------------------------------*/
    /*     PTF HAS BEEN SUCCESSFULLY PROCESSED                         */
    /*-----------------------------------------------------------------*/
       QSYS/SNDPGMMSG  MSGID(CPC1214) MSGF(*LIBL/QCPFMSG) +
                       MSGDTA(*NONE) TOPGMQ(*PRV (* *NONE +
                       *NONE)) TOMSGQ(*TOPGMQ) MSGTYPE(*COMP)
   RETURN
   /*-----------------------------------------------------------------*/
   /*     HANDLE ALL ERROR CONDITIONS                                 */
   /*-----------------------------------------------------------------*/
  HDLERR:
      /* Try to back out any changes already made */
      /* If nothing to back out or back-out operation was successful */
      QSYS/SNDPGMMSG  MSGID(CPF3638) MSGF(*LIBL/QCPFMSG) +
                      MSGDTA(*NONE) TOPGMQ(*PRV (* *NONE +
                      *NONE)) TOMSGQ(*TOPGMQ) MSGTYPE(*ESCAPE)
      /* Else the permanent changes not backed out    */
      QSYS/SNDPGMMSG  MSGID(CPF3639) MSGF(*LIBL/QCPFMSG) +
                      MSGDTA(*NONE) TOPGMQ(*PRV (* *NONE +
                      *NONE)) TOMSGQ(*TOPGMQ) MSGTYPE(*ESCAPE)

ENDPGM                              /* Return to external caller     */
```

Example: Creating an exit program for Operational Assistant backup

This example shows a user-written exit program, written in CL, for doing Operational Assistant backup.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
   /*****************************************************************/
   /*****************************************************************/
   /*                                                               */
   /* FUNCTION:  User-written exit program for doing Operational  */
   /*            Assistant backup.                                  */
   /*                                                               */
   /* LANGUAGE:  CL                                                 */
   /*                                                               */
   /* APIs USED: None                                               */
   /*                                                               */
   /*****************************************************************/
   /*****************************************************************/


   PGM   PARM(&PRODID &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
```

Machine interface programming

Machine interface (MI) is the interface or boundary between the operating system and the Licensed Internal Code. Use this information to develop, create, run, and debug an MI program.

While some MI instructions are discussed within the context of how to develop MI programs, the following information makes no attempt to review the full range of MI instructions. The goal is to provide a sufficient base of knowledge so that you can begin to use the MI language.

Related reference:

IBM i Machine Interface

Machine interface instructions

Machine interface (MI) instructions define the operations to be performed by an MI program.

Programs and procedures are the two basic units of execution on the System i® product. Programs come in two flavors: the original program model (OPM) and the Integrated Language Environment (ILE). MI programs can be created only for the OPM environment. If you require ILE support in the development of your applications, use ILE C, ILE COBOL, ILE CL, or ILE RPG and the built-in MI support provided by the languages.

In the OPM environment, a program consists of two basic components: the object definition table (ODT) and an instruction stream. MI programs are created by using the Create Program (QPRCRTPG) API.

The ODT is the means for defining all objects (program data elements) that are referred to by the MI instruction stream. An ODT definition of an object does not actually allocate storage for the object. It does, however, define when and how much storage is to be allocated and also the attributes of the storage (for example, the data type of the object). The ODT is built from the declare (DCL) statements found in the source used to create a program. Because DCL statements are actually instructions to the QPRCRTPG API and not MI instructions, they are defined in the QPRCRTPG API.

The following types of objects can be declared:

- Scalar
- Pointer
- Machine space pointer
- Operand list
- Instruction definition list
- Exception description
- Space
- Constant

The instruction stream defines the set of operations to be performed by the program. The instruction stream is built from the MI instructions found in the source used to create a program. The various MI instructions that you can use are defined in the machine interface information.

Within the source used to create a program, there is a type of statement called a directive. Directive statements can be found in the QPRCRTPG API and are used to do the following:

- Control the formatting of the output listing, such as the title, page ejection, and so on.
- Define entry points within the program for external and internal calls.
- Define breakpoints within the program to associate a breakpoint name to a particular MI instruction.
- Specify the end of the program source.

The program end (PEND) directive must be the last statement in the source, and it functions as a return external (RTX) MI instruction if logically processed as part of the instruction stream.

Noncomment source statements (declares, instructions, and directives) are always ended by a semicolon (`;`). Comments always begin with a slash and asterisk (`/*`) and end with an asterisk and slash (`*/`).

Related reference:

Create Program (QPRCRTPG) API

IBM i Machine Interface

Example: Writing an MI program

This example shows how to write a simple MI program that receives two packed-decimal parameters and returns the larger value through a third parameter.

This MI program demonstrates how to perform the following operations:

- Define an external entry point.
- Define and access parameters.
- Use conditional branching.
- Assign a value to a scalar object.
- End the program.

Setting the entry point

First the program, MI01 in this example, needs an ENTRY directive statement to designate its external entry point. The following directive declares an unnamed (the *) external (the EXT) entry point, which is called with a parameter list corresponding to PARM_LIST (defined later in the source code):

```
ENTRY * (PARM_LIST) EXT;
```

Setting the declare statements

IBM i programs typically pass parameters by reference as part of the high-level language (HLL) calling convention. Because IBM i programs pass by reference (that is, address and not value), the program also needs to define three space pointers (how storage is referenced) to represent the three parameters being passed. This is accomplished by the following directives:

```
DCL     SPCPTR      ARG1@      PARM;
DCL     SPCPTR      ARG2@      PARM;
DCL     SPCPTR      RESULT@    PARM;
```

To associate these three space pointers with the parameters being passed to the program, the following operand list (OL) is declared:

```
DCL     OL          PARM_LIST         /* Name of OL is PARM_LIST  */
                   (ARG1@,            /* The first parameter      */
                    ARG2@,            /* The second parameter     */
                    RESULT@)          /* The third parameter      */
                    PARM       EXT;   /* External parameter list  */
```

The names ARG1@, ARG2@, RESULT@, and PARM_LIST are chosen by you and are not mandated by the system. You can choose any valid name for any object data element. For a definition of what constitutes a valid name, see "Name" in the Program syntax topic of the Create Program (QPRCRTPG) API.

Now that the program has established addressability (the space pointers) to the three parameters, the program needs to declare how to map (or view) the storage addressed. The following declarations define the storage addressed (the BAS argument) by the three space pointer parameters as being packed-decimal (PKD) scalar data objects (DD) with 15 digits, 5 digits being to the right of the decimal point:

```
DCL     DD          ARG1       PKD(15,5)     BAS(ARG1@);
DCL     DD          ARG2       PKD(15,5)     BAS(ARG2@);
DCL     DD          RESULT     PKD(15,5)     BAS(RESULT@);
```

The names ARG1, ARG2, and RESULT are chosen arbitrarily, but, for ease of reading, are similar to the basing space pointers ARG1@, ARG2@, and RESULT@. The declarations of packed 15,5 are used for consistency with CL. The declared type and size could be of any other valid type and size. The true requirement is that the calling program and the MI program agree on the type and size.

Starting the instruction stream

With all the needed declarations now done, the instruction stream definition, where the program will compare the numeric values (CMPNV instruction) of parameters one and two, is started:

```
        CMPNV(B)    ARG1,ARG2  / LO(ITS2);
```

The program then branches (the (B) extender to CMPNV) to label ITS2 if ARG1 is less than ARG2 (the /LO branch target).

Note: MI instructions, such as CMPNV, are defined in IBM i Machine Interface. Pervasive instruction extenders, such as branch (B) and target keywords (LO, HI, EQ, and so on), are defined under Instruction Statement, which is a subheading in the Program Syntax section of the Create Program (QPRCRTPG) API topic.

If ARG1 is not low (LO) when compared to ARG2, the next MI instruction in the source stream is run. When the next MI instruction is run, it copies the numeric value (CPYNV instruction) of ARG1 to RESULT and, following that, branches to label RETURN:

```
        CPYNV       RESULT,ARG1;
        B           RETURN;
```

If ARG2 was greater than ARG1, the CPYNV instruction at label ITS2 is run, setting RESULT to the value of ARG2:

```
ITS2:   CPYNV       RESULT,ARG2;
```

The program has now finished processing and ends:

```
RETURN: RTX         *;
        PEND;
```

The previous return external (RTX) instruction is not needed because it is implied by the PEND directive. The RTX instruction is included to add clarity to the program flow.

MI01 program complete code example

Put all together, the program looks like this:

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: MI01                                            */
/*                                                                   */
/*     Programming Language: MI                                      */
/*                                                                   */
/*     Description: Return the larger of two packed arguments.       */
/*                                                                   */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
ENTRY * (PARM_LIST) EXT;
DCL     SPCPTR      ARG1@      PARM;
DCL     SPCPTR      ARG2@      PARM;
DCL     SPCPTR      RESULT@    PARM;
DCL     OL          PARM_LIST
                   (ARG1@,
                    ARG2@,
                    RESULT@)
                    PARM       EXT;
DCL     DD          ARG1       PKD(15,5)     BAS(ARG1@);
DCL     DD          ARG2       PKD(15,5)     BAS(ARG2@);
DCL     DD          RESULT     PKD(15,5)     BAS(RESULT@);
        CMPNV(B)    ARG1,ARG2  / LO(ITS2);
        CPYNV       RESULT,ARG1;
        B           RETURN;
ITS2:   CPYNV       RESULT,ARG2;
RETURN: RTX         *;
        PEND;
```

Compiling an MI program

If you enter the source into a source physical file, you can now compile the source and create an MI program. To create the program, use the Create Program (QPRCRTPG) API. Test and debug the program if it has errors. You can also declare an exception handler for the program.

Notes:

- The QPRCRTPG API assumes that the source statements presented to it are in code page 37. See IBM i Machine Interface for the specific code points that are required to build MI programs.
- By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Using CLCRTPG to create an MI program

Assume that the source is in a member named MI01 in the source file MISRC, which is created with a default record length (RCDLEN) of 92. The following CLCRTPG CL program can be used to create an MI program called MI01. (An MI program to call the Create Program (QPRCRTPG) API is developed in Creating an MI version of CLCRTPG.)

Note: All non-MI source examples are provided in CL, because CL is the one language (other than REXX) that is standard on all systems. Other high-level languages (HLLs) could be used in place of the CL programs (and in many cases would have been easier).

The following program reads a source file member into a program variable (&MIPGMSRC) and then does a CALL to the QPRCRTPG API. This program has many limitations (the major limitation is a program variable-size limit of 2000 bytes for the source), but provides for a reasonably simple MI program creation scenario.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: CLCRTPG                                         */
/*                                                                   */
/*     Programming Language: CL                                      */
/*                                                                   */
/*     Description: Create an MI program using the QPRCRTPG API.    */
/*                                                                   */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
             PGM        PARM(&SRCMBR)
             DCLF       FILE(MISRC)
             DCL        VAR(&SRCMBR) TYPE(*CHAR) LEN(10)
             DCL        VAR(&MIPGMSRC) TYPE(*CHAR) LEN(2000)
             DCL        VAR(&MIPGMSRCSZ) TYPE(*CHAR) LEN(4)
             DCL        VAR(&OFFSET) TYPE(*DEC) LEN(5 0) VALUE(1)
             DCL        VAR(&PGMNAM) TYPE(*CHAR) LEN(20) +
                          VALUE('          *CURLIB   ')
             DCL        VAR(&PGMTXT) TYPE(*CHAR) LEN(50) +
                          VALUE('Compare two packed arguments and +
                          return larger')
             DCL        VAR(&PGMSRCF) TYPE(*CHAR) LEN(20) +
                          VALUE('*NONE')
             DCL        VAR(&PGMSRCM) TYPE(*CHAR) LEN(10) VALUE(' ')
             DCL        VAR(&PGMSRCCHG) TYPE(*CHAR) LEN(13) VALUE(' ')
             DCL        VAR(&PRTFNAM) TYPE(*CHAR) LEN(20) +
                          VALUE('QSYSPRT   *LIBL     ')
             DCL        VAR(&PRTSTRPAG) TYPE(*CHAR) LEN(4) +
                          VALUE(X'00000001')
             DCL        VAR(&PGMPUBAUT) TYPE(*CHAR) LEN(10) +
                          VALUE('*ALL      ')
             DCL        VAR(&PGMOPTS) TYPE(*CHAR) LEN(22) +
                          VALUE('*LIST      *REPLACE   ')
             DCL        VAR(&NUMOPTS) TYPE(*CHAR) LEN(4) +
                          VALUE(X'00000002')
 LOOP:       RCVF
             MONMSG     MSGID(CPF0864) EXEC(GOTO CMDLBL(CRTPGM))
             CHGVAR     VAR(%SST(&MIPGMSRC &OFFSET 80)) VALUE(&SRCDTA)
             CHGVAR     VAR(&OFFSET) VALUE(&OFFSET + 80)
             GOTO       CMDLBL(LOOP)
 CRTPGM:     CHGVAR     VAR(%SST(&PGMNAM 1 10)) VALUE(&SRCMBR)
             CHGVAR     VAR(%BIN(&MIPGMSRCSZ)) VALUE(&OFFSET)
             CALL       PGM(QSYS/QPRCRTPG) PARM(&MIPGMSRC +
                          &MIPGMSRCSZ &PGMNAM &PGMTXT &PGMSRCF +
                          &PGMSRCM &PGMSRCCHG &PRTFNAM &PRTSTRPAG +
                          &PGMPUBAUT &PGMOPTS &NUMOPTS)
             ENDPGM
```

Creating the MI example program

After creating the CL program (assumed to be called CLCRTPG), the following statements create the previous MI program MI01:

```
DLTOVR MISRC
OVRDBF MISRC MBR(MI01)
CALL CLCRTPG MI01
```

Note: If the creation of MI01 fails, you should closely compare your source to that shown in this chapter. In general, consider the QPRCRTPG error messages that refer to "probable compiler error" as referring to your input source and not that the QPRCRTPG API itself is in error. (QPRCRTPG assumes its input is probably from a high-level language (HLL) compiler.)

Further, if the error message is CPF6399 (Identifier not declared), you can get an object definition table (ODT) listing by adding *XREF to the option template parameter (variable &PGMOPTS in the CLCRTPG program) when calling the QPRCRTPG API. Add *XREF to the existing *LIST and *REPLACE options, and change the number of option template entries parameter (variable &NUMOPTS) to 3.

Testing MI01

In this topic, assume that MI01 was successfully created.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: CL01                                            */
/*                                                                   */
/*     Programming Language: CL                                      */
/*                                                                   */
/*     Description: Test the MI program MI01.                        */
/*                                                                   */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
             PGM        PARM(&ARG1 &ARG2)
             DCL        VAR(&ARG1) TYPE(*DEC) LEN(15 5)
             DCL        VAR(&ARG2) TYPE(*DEC) LEN(15 5)
             DCL        VAR(&RESULT) TYPE(*DEC) LEN(15 5)
             DCL        VAR(&MSG) TYPE(*CHAR) LEN(20)
             DCL        VAR(&USR) TYPE(*CHAR) LEN(10)
             RTVJOBA    USER(&USR)
             CALL       PGM(MI01) PARM(&ARG1 &ARG2 &RESULT)
             CHGVAR     VAR(&MSG) VALUE(&RESULT)
             SNDMSG     MSG(&MSG) TOUSR(&USR)
             ENDPGM
```

The following statement calls the CL01 program:

```
CALL CL01 (-5    6)
```

This test should cause a message to be sent to your user message queue with the following value:

```
00000000000006.00000
```

Debugging the MI program

The MI program (MI01) that you created is a standard *PGM object. As you would expect, you can call MI01 from other high-level languages. You can delete MI01 with the Delete Program (DLTPGM) command, save and restore MI01 using the standard save (SAV) and restore (RST) commands, and so on.

You can also debug it using the standard debugger on the system. To debug it, you need to look at the listing produced by the QPRCRTPG API to determine the MI instruction number. Then use that number with the Add Breakpoint (ADDBKP) CL command. For example, when creating MI01 in the previous exercise, the following listing was generated by QPRCRTPG:

```
xxxxSS1 VxRxMx yymmdd                                Generated Output                                08/08/94 09:46:36  Page     1
  SEQ (1)INST Offset    Generated Code     *... ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 ... ... 8
 00001                                     ENTRY * (PARM_LIST) EXT                                                    ;
 00002                                     DCL SPCPTR ARG1@ PARM                                                      ;
 00003                                     DCL SPCPTR ARG2@ PARM                                                      ;
 00004                                     DCL SPCPTR RESULT@ PARM                                                    ;
 00005                                     DCL OL PARM_LIST (ARG1@, ARG2@, RESULT@) PARM EXT                          ;
 00006                                     DCL DD ARG1 PKD(15,5) BAS(ARG1@)                                           ;
 00007                                     DCL DD ARG2 PKD(15,5) BAS(ARG2@)                                           ;
 00008                                     DCL DD RESULT PKD(15,5) BAS(RESULT@)                                       ;
 00009  0001 000004  3C46 2000 0006 0007    CMPNV(B) ARG1,ARG2 / LO(ITS2)                                              ;
                    0009
 00010  0002 00000E  1042 0008 0006         CPYNV RESULT,ARG1                                                          ;
 00011  0003 000014  1011 000A              B RETURN                                                                   ;
 00012  0004 000018  3042 0008 0007  ITS2:  CPYNV RESULT,ARG2                                                          ;
 00013  0005 00001E  22A1 0000  (2)  RETURN: RTX *                                                                     ;
 00014  0006 000022  0260                   PEND                                                                       ;
 xxxxSS1 VxRxMx yymmdd                        Generated Output                                08/08/94 09:46:36  Page     2
  MSGID    ODT   ODT Name                                  Semantics and ODT Syntax Diagnostics
 xxxxSS1 VxRxMx yymmdd                        Generated Output                                08/08/94 09:46:36  Page     3
   MSGID   MI Instruction Stream Semantic Diagnostics
```

Setting breakpoints in the MI program

To view the value of RESULT at label RETURN, you first determine that RETURN corresponds to MI instruction (**(1)**) 0005 (**(2)**) and enter the following CL commands:

```
STRDBG PGM(MI01)
ADDBKP STMT('/0005') PGMVAR((RESULT ()))
CALL CL01 (-5    6)
```

The following display is shown:

```
+------------------------------------------------------------------------------+
|                                                                              |
|                              Display Breakpoint                              |
|                                                                              |
|  Statement/Instruction . . . . . . . . . : /0005                             |
|  Program . . . . . . . . . . . . . . . . : MI01                              |
|  Recursion level . . . . . . . . . . . . : 1                                 |
|  Start position  . . . . . . . . . . . . : 1                                 |
|  Format  . . . . . . . . . . . . . . . . : *CHAR                             |
|  Length  . . . . . . . . . . . . . . . . : *DCL                              |
|                                                                              |
|  Variable  . . . . . . . . . . . . . . . : RESULT                            |
|    Type  . . . . . . . . . . . . . . . . :   PACKED                          |
|    Length  . . . . . . . . . . . . . . . :   15 5                            |
|   '          6.00000'                                                        |
|                                                                              |
+------------------------------------------------------------------------------+
```

Breakpoints can also be set with a directive statement. Given that the MI01 program is able to be debugged and a break directive was not used, the purpose for which you use the directive may not be obvious. As mentioned in “Creating the MI example program” on page 501, many expected users of the QPRCRTPG API are compilers of HLLs. The break (BRK) directive allows users of the QPRCRTPG API to associate an HLL statement identifier with a generated MI instruction. For example, assume that MI01 was developed to be an implementation of a fictional HLL statement such as:

```
RESULT = MAX(ARG1, ARG2)
```

This assigns the MAX (defined as the largest argument) of ARG1 or ARG2 to RESULT. Also assume that an HLL programmer had written a program called HLLEXAMPLE with the following statements:

```
00001  RESULT = MAX(ARG1, ARG2)
00002  EXIT
```

By using break (BRK) directives, the QPRCRTPG user or compiler could associate the HLL statements with the generated MI instructions in the following way.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: MI01                                            */
/*                                                                   */
/*     Programming Language: MI                                      */
/*                                                                   */
/*     Description: Demonstrate how to associate HLL statement       */
/*                  identifiers with MI instructions using BRK       */
/*                  directives.                                      */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
ENTRY * (PARM_LIST) EXT;
DCL     SPCPTR      ARG1@      PARM;
DCL     SPCPTR      ARG2@      PARM;
DCL     SPCPTR      RESULT@    PARM;
DCL     OL          PARM_LIST
                   (ARG1@,
                    ARG2@,
                    RESULT@)
                    PARM       EXT;
DCL     DD          ARG1       PKD(15,5)     BAS(ARG1@);
DCL     DD          ARG2       PKD(15,5)     BAS(ARG2@);
DCL     DD          RESULT     PKD(15,5)     BAS(RESULT@);
BRK "00001";
        CMPNV(B)    ARG1,ARG2  / LO(ITS2);
        CPYNV       RESULT,ARG1;
        B           RETURN;
ITS2:   CPYNV       RESULT,ARG2;
BRK "00002";
RETURN: RTX         *;
        PEND;
```

This allows the HLL programmer to use the following to debug the HLL program by using the statement identifiers of the HLL:

```
STRDBG PGM(HLLEXAMPLE)
ADDBKP STMT(00002) PGMVAR((RESULT ()))
```

The following display shows that the HLL statement 00002 has been equated with MI instruction 0005 due to the use of BRK directives:

```
+------------------------------------------------------------------------------+
|                                                                              |
|                                                                              |
|                                                                              |
|                             Display Breakpoint                               |
|                                                                              |
|  Statement/Instruction . . . . . : 00002 /0005                               |
|  Program . . . . . : HLLEXAMPLE                                              |
|  Recursion level . . . . . : 1                                               |
|  Start position  . . . . . : 1                                               |
|  Format  . . . . . : *CHAR                                                   |
|  Length  . . . . . : *DCL                                                    |
|                                                                              |
|  Variable  . . . . . : RESULT                                                |
|    Type  . . . . . :   PACKED                                                |
|    Length  . . . . . :   15 5                                                |
|   '          6.00000'                                                        |
|                                                                              |
+------------------------------------------------------------------------------+
```

Handling exceptions in the MI program

As coded, the MI01 program works fine when it is passed packed decimal parameters. But when the MI01 program is passed other data types, such as in CALL CL01 (abc 6), exceptions occur. To handle these exceptions, additional statements could be added to MI01 so that:

- A 1-character return code parameter returns a status where `0` indicates no error and `1` indicates an error occurred.
- An exception description is defined to handle MCH1202 decimal data errors.

Add the following statements to MI01:

1. Declare a fourth space parameter to receive the return code parameter:

```
DCL     SPCPTR      RC@         PARM;
```

2. Update the operand list directive for PARM_LIST:

```
DCL     OL          PARM_LIST
                   (ARG1@,
                    ARG2@,
                    RESULT@,
                    RC@)              /* the new parameter  */
                    PARM       EXT;
```

3. Declare the storage addressed by RC@ as a 1-byte character data element:

```
DCL     DD          RC          CHAR(1)       BAS(RC@);
```

4. Declare an exception handler for MCH1202. With this exception description, all occurrences of MCH1202 will cause an immediate (IMD) branch to label M1202.

```
DCL     EXCM        DATAERROR  EXCID(H'0C02') BP (M1202) IMD;
```

   Note: The EXCID is the hexadecimal representation of the message identifier string 1202 where 12 = X'0C' and 02 = X'02'. While most MCH errors follow this relationship of message ID string to hexadecimal EXCID, always see IBM i Machine Interface to determine what specific exception IDs can be signaled by a given MI statement.

5. Because label M1202 is being used to indicate an error, set the return code to 1 by using copy bytes left-justified and then end:

```
M1202:  CPYBLA      RC,'1';
        RTX         *;
        PEND;
```

   A more complete example of how to handle exceptions is provided in Handling exceptions in the MICRTPG2 program.

6. Because the non-M1202 path indicates that no error was detected, update the normal return path:

```
RETURN: CPYBLA      RC,'0';
```

7. Because M1202 was appended to the end of the MI01 source, remove the original MI01 PEND directive.

Here is an updated view of the MI01 program:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: MI01                                            */
/*                                                                   */
/*     Programming Language: MI                                      */
/*                                                                   */
/*     Description: Enhanced version of MI program MI01 that        */
/*                  demonstrates enabling an exception monitor.      */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
```

```
ENTRY * (PARM_LIST) EXT;
DCL     SPCPTR      ARG1@      PARM;
DCL     SPCPTR      ARG2@      PARM;
DCL     SPCPTR      RESULT@    PARM;
DCL     SPCPTR      RC@        PARM;
DCL     OL          PARM_LIST
                   (ARG1@,
                    ARG2@,
                    RESULT@,
                    RC@)
                    PARM       EXT;
DCL     DD          ARG1       PKD(15,5)     BAS(ARG1@);
DCL     DD          ARG2       PKD(15,5)     BAS(ARG2@);
DCL     DD          RESULT     PKD(15,5)     BAS(RESULT@);
DCL     DD          RC         CHAR(1)       BAS(RC@);
DCL     EXCM        DATAERROR  EXCID(H'0C02') BP (M1202) IMD;
        CMPNV(B)    ARG1,ARG2  / LO(ITS2);
        CPYNV       RESULT,ARG1;
        B           RETURN;
ITS2:   CPYNV       RESULT,ARG2;
RETURN: CPYBLA      RC,'0';
        RTX         *;
M1202:  CPYBLA      RC,'1';
        RTX         *;
        PEND;
```

The following example updates CL01 to support the new return code parameter:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: CL01                                            */
/*                                                                   */
/*     Programming Language: CL                                      */
/*                                                                   */
/*     Description: Enhanced version of CL program CL01 that        */
/*                  demonstrates the use of enhanced MI01.           */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
             PGM        PARM(&ARG1 &ARG2)
             DCL        VAR(&ARG1) TYPE(*DEC) LEN(15 5)
             DCL        VAR(&ARG2) TYPE(*DEC) LEN(15 5)
             DCL        VAR(&RESULT) TYPE(*DEC) LEN(15 5)
             DCL        VAR(&RC) TYPE(*CHAR) LEN(1)
             DCL        VAR(&MSG) TYPE(*CHAR) LEN(20)
             DCL        VAR(&USR) TYPE(*CHAR) LEN(10)
             RTVJOBA    USER(&USR)
             CALL       PGM(MI01) PARM(&ARG1 &ARG2 &RESULT &RC)
             IF         COND(&RC = '0') +
               THEN(CHGVAR     VAR(&MSG) VALUE(&RESULT))
             ELSE +
               CHGVAR     VAR(&MSG) VALUE('ERROR FOUND')
             SNDMSG     MSG(&MSG) TOUSR(&USR)
             ENDPGM
```

After recompiling the MI01 program and the CL01 program, CALL CL01 (abc 6) now results in the following message (not the previous MCH1202):

```
ERROR FOUND
```

Related reference:

Create Program (QPRCRTPG) API

"Creating an MI version of the CLCRTPG program" on page 507
This topic discusses how to create an MI version of the CLCRTPG program that can be used to create MI

programs. This program is called MICRTPG.

"Creating the MICRTPG2 program" on page 521
This topic shows how to create the MICRTPG2 program and how to handle exceptions in the program.

Creating an MI version of the CLCRTPG program

This topic discusses how to create an MI version of the CLCRTPG program that can be used to create MI programs. This program is called MICRTPG.

Because the CLCRTPG program is used to create the initial version of MICRTPG and CLCRTPG can support only as many as 2000 bytes of source in the &MIPGMSRC variable, MICRTPG is initially defined with a minimal set of function. Significant additions to the MICRTPG program can be made after it is used as a building block in the creation of MI programs.

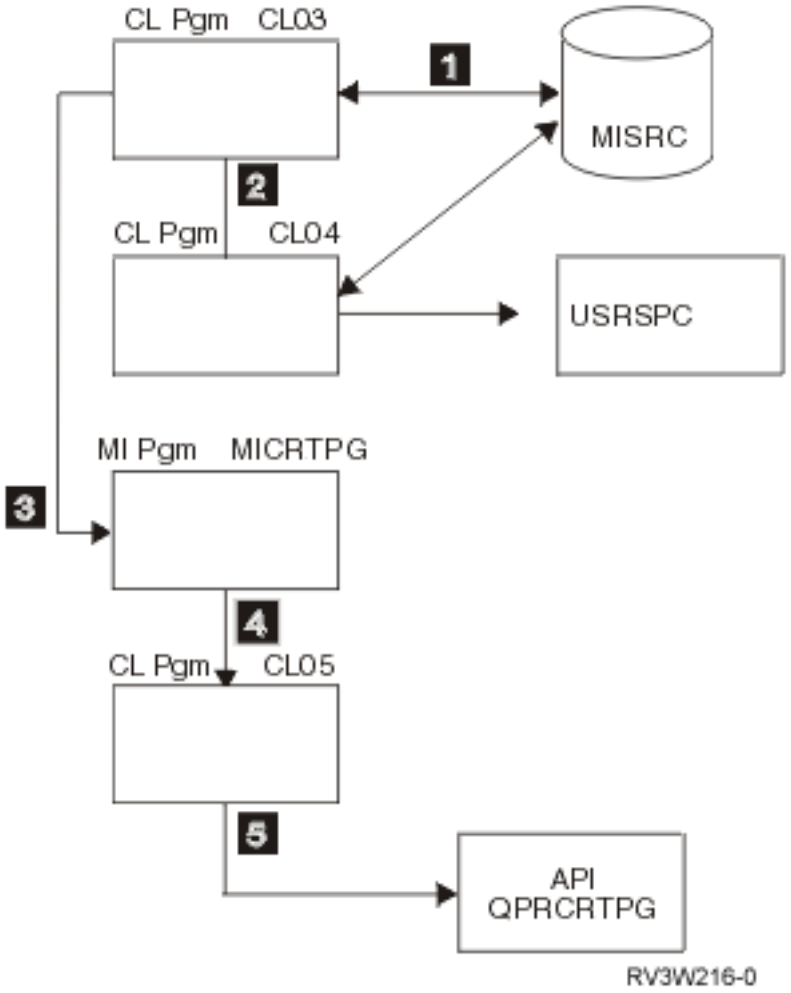
In the initial design (see the program flow in "Source for the CL03 program" on page 508), there are four programs. The first program is a CL program (CL03) that does the following:

- Creates a user space (*USRSPC) object of 64KB size to hold the MI source.
- Overrides the MISRC file to the appropriate source physical file and member **(1)**.
- Calls a second CL program (CL04), which loads the selected MISRC member into the user space (*USRSPC) **(2)**.
- Calls an MI program (MICRTPG) **(3)**. The MICRTPG program calls CL program CL05 **(4)** and passes addressability to the *USRSPC, where CL05 then calls the QPRCRTPG API **(5)**.

The MICRTPG program demonstrates how to perform the following operations:

- Define a structure.
- Initialize declared storage
- Use two different approaches to resolve a system pointer to an external object.
- Assign a space pointer to address a user space.
- Call a program and pass three parameters.

The overall program flow for creating the MICRTPG program appears as follows:



Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Source for the CL03 program

The source for CL03 follows:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: CL03                                            */
/*                                                                   */
/*     Programming Language: CL                                      */
/*                                                                   */
/*     Description: Main driver program for initial version of       */
/*                  MI program MICRTPG.  This program creates a      */
/*                  *USRSPC, calls CL04 to load MI source from       */
/*                  a *SRC physical file into the *USRSPC, and       */
/*                  then calls MICRTPG to create MI programs.        */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
             PGM        PARM(&FILE &MBR)
             DCL        VAR(&FILE) TYPE(*CHAR) LEN(10)
             DCL        VAR(&MBR) TYPE(*CHAR) LEN(10)
```

```
            DCL        VAR(&SPCNAM) TYPE(*CHAR) LEN(20) +
                         VALUE('          *CURLIB   ')
            DCL        VAR(&SPCEXTATR) TYPE(*CHAR) LEN(10) VALUE(' ')
            DCL        VAR(&SPCSIZ) TYPE(*CHAR) LEN(4) +
                         VALUE(X'00010000')
            DCL        VAR(&SPCINTVAL) TYPE(*CHAR) LEN(1) VALUE(X'00')
            DCL        VAR(&SPCSPCAUT) TYPE(*CHAR) LEN(10) +
                         VALUE('*ALL')
            DCL        VAR(&SPCTXTDSC) TYPE(*CHAR) LEN(50) VALUE(' ')
            DCL        VAR(&SPCRPLOPT) TYPE(*CHAR) LEN(10) +
                         VALUE('*YES')
            DCL        VAR(&ERRCOD) TYPE(*CHAR) LEN(4) +
                         VALUE(X'00000000')
            DCL        VAR(&SPCDMN) TYPE(*CHAR) LEN(10) VALUE('*USER')
            DCL        VAR(&BINOFFSET) TYPE(*CHAR) LEN(4) +
                         VALUE(X'00000001')
            CHGVAR     VAR(%SST(&SPCNAM 1 10)) VALUE(&MBR)
            CALL       PGM(QUSCRTUS) PARM(&SPCNAM &SPCEXTATR +
                         &SPCSIZ &SPCINTVAL &SPCSPCAUT &SPCTXTDSC +
                         &SPCRPLOPT &ERRCOD &SPCDMN)
            OVRDBF     FILE(MISRC) TOFILE(&FILE) MBR(&MBR)
            CALL       PGM(CL04) PARM(&MBR &BINOFFSET)
            CALL       PGM(MICRTPG) PARM(&MBR &BINOFFSET)
            ENDPGM
```

Source for the CL04 program

The source for CL04 follows:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: CL04                                            */
/*                                                                   */
/*     Programming Language: CL                                      */
/*                                                                   */
/*     Description: Load a source physical file member into the      */
/*                  *USRSPC named &MBR.                              */
/*                                                                   */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
            PGM        PARM(&MBR &BINOFFSET)
            DCLF       FILE(MISRC)
            DCL        VAR(&MBR) TYPE(*CHAR) LEN(10)
            DCL        VAR(&BINOFFSET) TYPE(*CHAR) LEN(4)
            DCL        VAR(&OFFSET) TYPE(*DEC) LEN(8 0) VALUE(1)
            DCL        VAR(&LENGTH) TYPE(*CHAR) LEN(4) +
                         VALUE(X'00000050')
            DCL        VAR(&SPCNAM) TYPE(*CHAR) LEN(20) +
                         VALUE('          *LIBL     ')
            CHGVAR     VAR(%SST(&SPCNAM 1 10)) VALUE(&MBR)
LOOP:       RCVF
            MONMSG     MSGID(CPF0864) EXEC(GOTO CMDLBL(DONE))
            CALL       PGM(QUSCHGUS) PARM(&SPCNAM &BINOFFSET +
                         &LENGTH &SRCDTA '0')
            CHGVAR     VAR(&OFFSET) VALUE(&OFFSET + 80)
            CHGVAR     VAR(%BIN(&BINOFFSET)) VALUE(&OFFSET)
            GOTO       CMDLBL(LOOP)
DONE:       ENDPGM
```

Source for the CL05 program

The source for CL05 follows:

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*     Program Name: CL05                                        */
/*                                                               */
/*     Programming Language: CL                                  */
/*                                                               */
/*     Description: Create an MI program using the QPRCRTPG API. */
/*                                                               */
/*                                                               */
/*     Header Files Included: None                               */
/*                                                               */
/*                                                               */
/*****************************************************************/
             PGM        PARM(&SRCMBR &MIPGMSRC &MIPGMSRCSZ)
             DCL        VAR(&SRCMBR) TYPE(*CHAR) LEN(10)
             DCL        VAR(&MIPGMSRC) TYPE(*CHAR) LEN(1)
             DCL        VAR(&MIPGMSRCSZ) TYPE(*CHAR) LEN(4)
             DCL        VAR(&PGMNAM) TYPE(*CHAR) LEN(20) +
                          VALUE('          *CURLIB   ')
             DCL        VAR(&PGMTXT) TYPE(*CHAR) LEN(50) +
                          VALUE(' ')
             DCL        VAR(&PGMSRCF) TYPE(*CHAR) LEN(20) +
                          VALUE('*NONE')
             DCL        VAR(&PGMSRCM) TYPE(*CHAR) LEN(10) VALUE(' ')
             DCL        VAR(&PGMSRCCHG) TYPE(*CHAR) LEN(13) VALUE(' ')
             DCL        VAR(&PRTFNAM) TYPE(*CHAR) LEN(20) +
                          VALUE('QSYSPRT   *LIBL     ')
             DCL        VAR(&PRTSTRPAG) TYPE(*CHAR) LEN(4) +
                          VALUE(X'00000001')
             DCL        VAR(&PGMPUBAUT) TYPE(*CHAR) LEN(10) +
                          VALUE('*ALL      ')
             DCL        VAR(&PGMOPTS) TYPE(*CHAR) LEN(22) +
                          VALUE('*LIST      *REPLACE   ')
             DCL        VAR(&NUMOPTS) TYPE(*CHAR) LEN(4) +
                          VALUE(X'00000002')
             CHGVAR     VAR(%SST(&PGMNAM 1 10)) VALUE(&SRCMBR)
             CALL       PGM(QSYS/QPRCRTPG) PARM(&MIPGMSRC +
                          &MIPGMSRCSZ &PGMNAM &PGMTXT &PGMSRCF +
                          &PGMSRCM &PGMSRCCHG &PRTFNAM &PRTSTRPAG +
                          &PGMPUBAUT &PGMOPTS &NUMOPTS)
             ENDPGM
```

Source for the MICRTPG program

The source for MICRTPG follows:

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*     Program Name: MICRTPG                                     */
/*                                                               */
/*     Programming Language: MI                                  */
/*                                                               */
/*     Description: Initial version of MI program to create      */
/*                  additional MI programs using the QPRCRTPG API. */
/*                                                               */
/*                                                               */
/*     Header Files Included: None                               */
/*                                                               */
/*                                                               */
/*****************************************************************/
ENTRY * (PARM_LIST) EXT;
DCL SPCPTR MBR@ PARM;
DCL SPCPTR BINOFFSET@ PARM;
DCL OL PARM_LIST (MBR@, BINOFFSET@) PARM EXT;
DCL DD MBR CHAR(10) BAS(MBR@);
```

```
DCL DD BINOFFSET BIN(4) BAS(BINOFFSET@);
DCL DD RSLVOBJ CHAR(34);
 DCL DD RSLVTYPE CHAR(1) DEF(RSLVOBJ) POS(1) INIT(X'19');
 DCL DD RSLVSUBTYPE CHAR(1) DEF(RSLVOBJ) POS(2) INIT(X'34');
 DCL DD RSLVNAME CHAR(30) DEF(RSLVOBJ) POS(3);
 DCL DD RSLVAUTH CHAR(2) DEF(RSLVOBJ) POS(33) INIT(X'0000');
DCL SYSPTR USRSPCOBJ;
DCL SPCPTR USRSPC;
DCL SYSPTR CL05 INIT("CL05", TYPE(PGM));
DCL OL CL05OL (MBR@, USRSPC, BINOFFSET@) ARG;
CPYBLAP RSLVNAME, MBR, ' ';
RSLVSP USRSPCOBJ, RSLVOBJ, *, *;
SETSPPFP USRSPC, USRSPCOBJ;
CALLX CL05, CL05OL, *;
RTX *;
PEND;
```

Writing the MICRTPG program (by sections of code)

You will recognize some of these statements from the MI01 example, but others are new.

The following statements, which you have seen, for example, in MI01 program complete code example, define the entry point to this program and the parameters being passed on the call:

```
ENTRY * (PARM_LIST) EXT;
DCL SPCPTR MBR@ PARM;
DCL SPCPTR BINOFFSET@ PARM;
DCL OL PARM_LIST (MBR@, BINOFFSET@) PARM EXT;
DCL DD MBR CHAR(10) BAS(MBR@);
DCL DD BINOFFSET BIN(4) BAS(BINOFFSET@);
```

Declaring the structure

The following, however, are new statements:

```
DCL DD RSLVOBJ CHAR(34);
 DCL DD RSLVTYPE CHAR(1) DEF(RSLVOBJ) POS(1) INIT(X'19');
 DCL DD RSLVSUBTYPE CHAR(1) DEF(RSLVOBJ) POS(2) INIT(X'34');
 DCL DD RSLVNAME CHAR(30) DEF(RSLVOBJ) POS(3);
 DCL DD RSLVAUTH CHAR(2) DEF(RSLVOBJ) POS(33) INIT(X'0000');
```

These statements declare a structure named RSLVOBJ that comprises four subelements defined within it. The subelements specify their position relative to the start of the structure RSLVOBJ. In the cases of the RSLVTYPE, RSLVSUBTYPE, and RSLVAUTH data elements, they initialize the associated storage.

The RSLVOBJ structure is used later in the program as input to the Resolve System Pointer (RSLVSP) MI instruction. The RSLVSP instruction resolves (establishes addressability) to a user space (*USRSPC) (the X'1934' object type and subtype) named RSLVNAME. The RSLVNAME user space is assigned from the source member name (MBR) data element. This user space is the one created in “Source for the CL03 program” on page 508. If you are interested in the details of this structure, see Resolve System Pointer (RSLVSP).

Note: In the declare (DCL) statement of RSLVOBJ, the leading blanks used to indent the subelements (for example, RSLVTYPE and RSLVSUBTYPE) are strictly to enhance the readability of the source. They are not a requirement of the QPRCRTPG API. In general, you can use strings of blanks of any length in the source of a program. Blanks, one or more, are simply used as delimiters in identifying tokens. The major exception is the INIT argument of a DCL statement where the number of blanks is important. For example, the previous declare statement could have been written as follows and other than readability, nothing would have been lost:

```
DCL DD RSLVOBJ CHAR(34); DCL DD RSLVTYPE CHAR(1)
DEF(RSLVOBJ) POS(1) INIT(X'19');        DCL DD RSLVSUBTYPE CHAR(1)
DEF(RSLVOBJ)                                  POS(2)
INIT(X'34'); DCL DD RSLVNAME CHAR(30) DEF(RSLVOBJ) POS(3); DCL
DD RSLVAUTH CHAR(2) DEF(RSLVOBJ) POS(33) INIT(X'0000');
```

Declaring pointers

The next statements declare a system pointer named USRSPCOBJ and a space pointer named USRSPC. USRSPCOBJ contains the address of the *USRSPC object after the execution of the RSLVSP instruction later in the instruction stream. USRSPC addresses the first byte of the *USRSPC:

```
DCL SYSPTR USRSPCOBJ;
DCL SPCPTR USRSPC;
```

Defining an external call

Because this program also uses the call external (CALLX) instruction to call the CL program CL05, define a system pointer for CL05:

```
DCL SYSPTR CL05 INIT("CL05", TYPE(PGM));
```

The preceding statement causes the QPRCRTPG API to initialize the system pointer CL05 to the name of the PGM CL05. The CL05 pointer is not set to the address of the CL05 object–this happens the first time the CL05 pointer is referred to in the instruction stream. If you review the declare statement in the QPRCRTPG API, notice that the context (CTX) argument uses the default. Using the context default (better known as library to most programmers) is equivalent to specifying *LIBL. *LIBL is referred to as the process name resolution list in the machine interface instructions.

Because this program calls the CL05 program (CALLX CL05) with parameters, it now defines an operand list CL05OL, which specifies the arguments to be passed on the CALLX:

```
DCL OL CL05OL (MBR@, USRSPC, BINOFFSET@) ARG;
```

When you get to the instruction stream of MICRTPG, copy the passed parameter MBR to the data structure element RSLVNAME. As RSLVNAME is defined as CHAR(30) and MBR is CHAR(10), the program uses the copy bytes left-justified with pad (CPYBLAP) instruction to set the rightmost 20 bytes of RSLVNAME to the value of the third argument (in this case, blanks):

```
CPYBLAP RSLVNAME, MBR, ' ';
```

Having established the *USRSPC name, use the RSLVSP instruction to get addressability to the object itself:

```
RSLVSP USRSPCOBJ, RSLVOBJ, *, *;
```

Note: Similar to how the *USRSPC name was resolved, RSLVSP could be used with a type of X'02' and a subtype of X'01' to resolve a system pointer to the CL05 *PGM object. The two different approaches were used to demonstrate the different styles (RSLVSP is clearly more flexible) and also to stay within the 2000-byte limit of the program source size imposed by the CLCRTPG program.

Then set the USRSPC space pointer to the first byte of the *USRSPC:

```
SETSPPFP USRSPC, USRSPCOBJ;
```

Calling the CL05 Program

Now the program will call the CL05 program (CALLX CL05) and pass the address of the *USRSPC as a parameter (along with the member name, program name, and the size of the source stream). When you call CL05 with the operand list CL05OL, CL05 passes the actual space pointer USRSPC. CL05 does not pass a space pointer that refers to the space pointer USRSPC (as opposed to how MBR@ and

BINOFFSET@ are passed to refer to MBR and BINOFFSET, respectively). This has the effect of having the CL05 program treat the *USRSPC storage as the parameter:

```
CALLX CL05, CL05OL, *;
```

Finally, as the program comes to an end, this is the return external instruction and pend directive for the initial version of MICRTPG:

```
RTX *;
PEND;
```

Creating the MICRTPG program

To create MICRTPG, use the following CL commands:

```
DLTOVR MISRC
OVRDBF MISRC MBR(MICRTPG)
CALL CLCRTPG MICRTPG
```

Assuming a successful creation, the CLCRTPG program is not used again because of the MI base with which to work (for example, MICRTPG is used as a boot-strap for further compiler enhancement).

Related tasks:

"Enhanced version of the MICRTPG program"
The enhanced version of the MICRTPG program (named MICRTPG2) incorporates the functions of the CL03 program and the CL05 program.

Related reference:

"Compiling an MI program" on page 500
If you enter the source into a source physical file, you can now compile the source and create an MI program. To create the program, use the Create Program (QPRCRTPG) API. Test and debug the program if it has errors. You can also declare an exception handler for the program.

"Example: Writing an MI program" on page 498
This example shows how to write a simple MI program that receives two packed-decimal parameters and returns the larger value through a third parameter.

IBM i Machine Interface

"Internal object types" on page 62
Internal objects are used to store the information needed to perform some system functions. The table shows the predefined values for all the IBM i internal object types.

Create Program (QPRCRTPG) API

Enhanced version of the MICRTPG program

The enhanced version of the MICRTPG program (named MICRTPG2) incorporates the functions of the CL03 program and the CL05 program.

A modified form of CL04 (renamed to CL06) is used in these examples to read the MISRC source physical file because MI instruction support for database access is beyond the scope of this topic collection.

The MICRTPG2 program demonstrates how to perform the following operations:

- Receive a variable number of parameters.
- Use static and automatic storage.
- Create a space object.
- Perform arithmetic operations.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Related reference:

"Creating an MI version of the CLCRTPG program" on page 507
This topic discusses how to create an MI version of the CLCRTPG program that can be used to create MI programs. This program is called MICRTPG.

Writing the MICRTPG2 program (by sections of code)

To write the MICRTPG2 program, follow these steps:

1. Define the entry point and associated parameters:

```
ENTRY * (PARM_LIST) EXT;
DCL SPCPTR FIL@ PARM;
DCL SPCPTR MBR@ PARM;
DCL OL PARM_LIST (MBR@, FIL@) PARM EXT MIN(1);
DCL DD FIL CHAR(10) BAS(FIL@);
DCL DD MBR CHAR(10) BAS(MBR@);
DCL DD NUM_PARMS BIN(4);
```

2. Have the MICRTPG2 program create an automatically extendable space (it can automatically increase to as many as 16 MB) using the Create Space (CRTS) instruction. Because the CRTS instruction requires a definition template, you need to define it. For details, see IBM i Machine Interface.

   The following template creates a space (type and subtype equal to X'19EF') that is defined through the OBJCRTOPT data element **(1)**. The space is defined as temporary (the next initial program load (IPL) will free up the storage occupied by the space), extendable up to as many as 16MB, and within a context (a library).

```
DCL DD CRTSTMPLT CHAR(160) BDRY(16);
 DCL DD TMPLTSPEC CHAR(8) DEF(CRTSTMPLT) POS(1);
  DCL DD TMPLTSIZE BIN(4) DEF(TMPLTSPEC) POS(1) INIT(160);
  DCL DD TMPLTBA   BIN(4) DEF(TMPLTSPEC) POS(5) INIT(0);
 DCL DD OBJID CHAR(32) DEF(CRTSTMPLT) POS(9);
  DCL DD SPCTYPE CHAR(1) DEF(OBJID) POS(1) INIT(X'19');
  DCL DD SPCSUBTYPE CHAR(1) DEF(OBJID) POS(2) INIT(X'EF');
  DCL DD SPCNAME CHAR(30) DEF(OBJID) POS(3) INIT(" ");
 DCL DD OBJCRTOPT CHAR(4) DEF(CRTSTMPLT) POS(41) INIT(X'60020000'); (1)
 DCL DD OBJRCVOPTS CHAR(4) DEF(CRTSTMPLT) POS(45);
  DCL DD * CHAR(2) DEF(OBJRCVOPTS) POS(1) INIT(X'0000');
  DCL DD ASP CHAR(2) DEF(OBJRCVOPTS) POS(3) INIT(X'0000');
 DCL DD SPCSIZ BIN(4) DEF(CRTSTMPLT) POS(49) INIT(1);
 DCL DD INTSPCVAL CHAR(1) DEF(CRTSTMPLT) POS(53) INIT(X'00');
 DCL DD PERFCLASS CHAR(4) DEF(CRTSTMPLT) POS(54) INIT(X'00000000');
 DCL DD * CHAR(1) DEF(CRTSTMPLT) POS(58) INIT(X'00');
 DCL DD PUBAUT CHAR(2) DEF(CRTSTMPLT) POS(59) INIT(X'0000');
 DCL DD TMPLTEXTN BIN(4) DEF(CRTSTMPLT) POS(61) INIT(96);
 DCL SYSPTR CONTEXT DEF(CRTSTMPLT) POS(65);
 DCL SYSPTR ACCESSGRP DEF(CRTSTMPLT) POS(81);
 DCL SYSPTR USRPRF DEF(CRTSTMPLT) POS(97);
 DCL DD MAXSPCSIZ BIN(4) DEF(CRTSTMPLT) POS(113) INIT(0);
 DCL DD DOMAIN CHAR(2) DEF(CRTSTMPLT) POS(117) INIT(X'0001');
 DCL DD * CHAR(42) DEF(CRTSTMPLT) POS(119) INIT((42)X'00');
```

3. Establish addressability to the CRTS template:

```
DCL SPCPTR CRTSTMPLT@ INIT(CRTSTMPLT);
```

4. Because the space is defined to be in a context, supply the address of the context in the previous CRTS template. This program uses the QTEMP context that is identified by the following:

```
DCL SYSPTR QTEMP@ BASPCO POS(65);
```

   Use the copy bytes with pointers instruction (CPYBWP) to set the template context data element.

```
CPYBWP CONTEXT, QTEMP@;
```

5. In the instruction stream, create the space:

```
CRTS USRSPC@, CRTSTMPLT@;
```

   This returns a system pointer to the created space in the system pointer:

```
DCL SYSPTR USRSPC@;
```

6. Declare a space pointer for addressability to the space through a space pointer (as opposed to the system pointer returned by the CRTS instruction):

   ```
   DCL SPCPTR USRSPC;
   ```

7. To keep track of how many bytes of source are loaded into the *USRSPC, define BINOFFSET. BINOFFSET is also being defined very specifically as an integer (BIN(4)) because it will be used later in the program with the set space pointer offset (SETSPPO) MI instruction. This requires an integer argument to refer to the space:

   ```
   DCL DD BINOFFSET BIN(4) AUTO INIT(0);
   ```

8. Because the size of the source is also a parameter to the QPRCRTPG API, define a space pointer to refer to BINOFFSET:

   ```
   DCL SPCPTR BINOFFSET@ AUTO INIT(BINOFFSET);
   ```

   The two previous declare statements have also introduced a new attribute to the DCL statement. Previously, all of the DCLs used the default of static (STAT) storage. BINOFFSET and BINOFFSET@, however, are being allocated from automatic (AUTO) storage. Many hours of debug time can be saved if you understand how the system manages these types of storage. For more information about the types of storage, see "Program storage" on page 529.

   So that the program does not retain the size of the source loaded from previous invocations of the program, you can declare BINOFFSET as being automatic. Because BINOFFSET@ needs to be set to the address of BINOFFSET (so that BINOFFSET can be passed as a parameter to CL06), you will also declare it as automatic. An alternative to using automatic storage would have been to explicitly set a static storage BINOFFSET to 0 by using CPYNV, but this does not allow for a discussion of the storage management differences.

9. Use the CL06 program to load the space after it is created. Because CL06 is limited to only 2000 bytes of addressability per parameter per call (CALLX), the MICRTPG2 program uses the Override with Database File (OVRDBF) CL command to cause the CL06 program to read and load twenty 80-byte source records per call. The source records are read starting at 1 on the first call, 21 on the second, 41 on the third, and so on. To run CL commands from the MICRTPG2 program, the program uses the Execute Command (QCMDEXC) API:

   ```
   DCL SYSPTR QCMDEXC INIT("QCMDEXC", CTX("QSYS"), TYPE(PGM));
   ```

10. Format the appropriate character strings for the Override with Database File (OVRDBF) CL command:

    Note: In the following declare (DCL) statement for CLOVRCMD, the 3 strings of '1234567890' are used strictly so that you can see that 10 bytes are being used. The strings themselves are overridden by the subsequent subelement DCLs for FILNAM, MBRNAM, and RECNUM, and could be replaced by 10 blanks:

    ```
    DCL DD CLOVRCMD CHAR(65);
     DCL DD OVRSTR CHAR(39) DEF(CLOVRCMD) POS(1)
         INIT("OVRDBF MISRC 1234567890 MBR(1234567890)");
     DCL DD OVRSTR2 CHAR(26) DEF(CLOVRCMD) POS(40)
         INIT(" POSITION(*RRN 1234567890)");
     DCL DD FILNAM CHAR(10) DEF(CLOVRCMD) POS(14);
     DCL DD MBRNAM CHAR(10) DEF(CLOVRCMD) POS(29);
     DCL DD RECNUM ZND(10,0) DEF(CLOVRCMD) POS(55);
    ```

11. Format the appropriate character strings for the Delete Override (DLTOVR) CL command. Because the OVRDBF commands are issued repetitively to progress through the source, the previous overrides need to be deleted:

    ```
    DCL DD CLDLTCMD CHAR(12) INIT("DLTOVR MISRC");
    ```

12. Establish space pointers to the CL command parameters, and, because the QCMDEXC API is being used, define the CL command string lengths as parameters:

```
DCL SPCPTR CLOVRCMD@ INIT(CLOVRCMD);
DCL SPCPTR CLDLTCMD@ INIT(CLDLTCMD);
DCL DD CLOVRLNG PKD(15,5) INIT(P'65');  /* Length of OVRDBF CL cmd */
DCL SPCPTR CLOVRLNG@ INIT(CLOVRLNG);
DCL DD CLDLTLNG PKD(15,5) INIT(P'12');  /* Length of DLTOVR CL cmd */
DCL SPCPTR CLDLTLNG@ INIT(CLDLTLNG);
```

13. Define the operand list (OL) definitions for calling the QCMDEXC API under the two different conditions:

```
DCL OL QCMDOVROL (CLOVRCMD@, CLOVRLNG@) ARG;
DCL OL QCMDDLTOL (CLDLTCMD@, CLDLTLNG@) ARG;
```

14. Because CALLX CL06 is called to load the space, declare its system pointer, parameters, and OL:

```
DCL SYSPTR CL06 INIT("CL06", TYPE(PGM));
DCL DD OFFSET PKD(15,5);
DCL SPCPTR OFFSET@ INIT(OFFSET);
DCL OL CL06OL (USRSPC, OFFSET@) ARG;
```

15. Declare the system pointer, parameters, and OL for the QPRCRTPG API:

```
DCL DD PGM CHAR(20);
 DCL DD PGMNAM CHAR(10) DEF(PGM) POS(1);
 DCL DD PGMLIBNAM CHAR(10) DEF(PGM) POS(11) INIT("*CURLIB   ");
DCL SPCPTR PGM@ INIT(PGM);
DCL DD PGMTXT CHAR(50) INIT(" ");
DCL SPCPTR PGMTXT@ INIT(PGMTXT);
DCL DD PGMSRCF CHAR(20) INIT("*NONE");
DCL SPCPTR PGMSRCF@ INIT(PGMSRCF);
DCL DD PGMSRCM CHAR(10) INIT(" ");
DCL SPCPTR PGMSRCM@ INIT(PGMSRCM);
DCL DD PGMSRCCHG CHAR(13) INIT(" ");
DCL SPCPTR PGMSRCCHG@ INIT(PGMSRCCHG);
DCL DD PRTFNAM CHAR(20) INIT("QSYSPRT   *LIBL     ");
DCL SPCPTR PRTFNAM@ INIT(PRTFNAM);
DCL DD PRTSTRPAG BIN(4) INIT(1);
DCL SPCPTR PRTSTRPAG@ INIT(PRTSTRPAG);
DCL DD PGMPUBAUT CHAR(10) INIT("*ALL      ");
DCL SPCPTR PGMPUBAUT@ INIT(PGMPUBAUT);
DCL DD PGMOPTS(16) CHAR(11) INIT((1)"*LIST", *(2)(1)"*REPLACE");
DCL SPCPTR PGMOPTS@ INIT(PGMOPTS);
DCL DD NUMOPTS BIN(4) INIT(2);
DCL SPCPTR NUMOPTS@ INIT(NUMOPTS);
DCL OL QPRCRTPGOL (USRSPC, BINOFFSET@, PGM@, PGMTXT@, PGMSRCF@,
                   PGMSRCM@, PGMSRCCHG@, PRTFNAM@, PRTSTRPAG@,
                   PGMPUBAUT@, PGMOPTS@, NUMOPTS@) ARG;
DCL SYSPTR QPRCRTPG INIT("QPRCRTPG", CTX("QSYS"), TYPE(PGM));
```

Beginning the instruction stream

Begin the instruction stream definition by doing the following:

1. Use the store parameter list length (STPLLEN) instruction to determine the number of parameters that were passed to the program:

```
STPLLEN NUM_PARMS;
```

2. If the number of parameters is 1, assign FILNAM to the value MISRC (the default that this program supports for the source physical file) and branch to label PARM1 to set the source member name:

```
CMPNV(B) NUM_PARMS, 2 / EQ(PARM2);
CPYBLAP FILNAM, 'MISRC', ' ';
B PARM1;
```

3. If the number of parameters is 2, assign FILNAM to the value of the second parameter:

```
PARM2: CPYBLA FILNAM, FIL;
```

4. Assign the source member name:

```
PARM1: CPYBLA MBRNAM, MBR;
```

5. Assign the proper context for the space:

```
CPYBWP CONTEXT, QTEMP@;
```

6. After establishing the context of the space, now create the space:

```
CRTS USRSPC@, CRTSTMPLT@;
```

7. Assign the space pointer USRSPC to address the first byte of the space:

```
SETSPPFP USRSPC, USRSPC@;
```

8. Set the OVRDBF CL command to start with POSITION(1):

```
CPYNV RECNUM, 1;
```

Using static storage to your advantage

In "Beginning the instruction stream" on page 516, the instructions can be done once and the space reused on subsequent invocations of the program. As a performance enhancement, add a check to see if this program has been previously called. To do the check, add a control field, and conditionally branch around the CRTS-oriented instructions if this call is not the initial call:

```
       STPLLEN NUM_PARMS;
       CMPNV(B) NUM_PARMS, 2 / EQ(PARM2);
       CPYBLAP FILNAM, 'MISRC', ' ';
       B PARM1;
PARM2: CPYBLA FILNAM, FIL;
PARM1: CPYBLA MBRNAM,MBR;
       CMPBLA(B) READY, '1' / EQ(SKIP);
       CPYBWP CONTEXT, QTEMP@;
       CRTS USRSPC@, CRTSTMPLT@;
       SETSPPFP USRSPC,USRSPC@;
       CPYBLA READY, '1';
SKIP:  CPYNV RECNUM, 1;
```

Resuming the program flow of the MICRTPG2 program from "Beginning the instruction stream" on page 516, you should have the program perform the following:

1. Fall into a loop (the MORE label) until all source records are loaded as the source physical file member position is overridden:

   ```
   MORE: CALLX QCMDEXC, QCMDOVROL, *;
   ```

2. Instruct the CL06 program to load source records from the start of the input buffer, which is actually the BINOFFSET into the space created earlier:

   ```
   CPYNV OFFSET,1;
   CALLX CL06, CL06OL, *;
   ```

3. Back out (subtract) the base-1 nature of CL using the short (the (S) extender) form of the subtract numeric (SUBN) instruction:

   ```
   SUBN(S) OFFSET, 1;
   ```

4. Add the number of MI source bytes processed by CL06 to the offset into the space (for the next call):

   ```
   ADDN(S) BINOFFSET, OFFSET;
   SETSPPO USRSPC, BINOFFSET;
   ```

5. Update the Override with Database File (OVRDBF) position parameter for the next call to CL06:

   ```
   ADDN(S) RECNUM, 20;
   ```

6. Delete the previous OVRDBF:

   ```
   CALLX QCMDEXC, QCMDDLTOL, *;
   ```

7. Check to see if all records were processed, and if not, branch to label MORE to load more source records:

   ```
   CMPNV(B) OFFSET, 1600 /EQ(MORE);
   ```

   Otherwise, assume that all source was loaded and prepare for calling the QPRCRTPG API by setting the program name:

   ```
   CPYBLA PGMNAM, MBR;
   ```

8. Reset the space pointer from the source of the input program to the start of the space. This resetting of the static storage USRSPC is also assumed in the branch to label SKIP earlier in the program:

   ```
   SETSPPO USRSPC,0;
   ```

9. Call the QPRCRTPG API to create the MI program:

```
CALLX QPRCRTPG, QPRCRTPGOL, *;
```

10. Indicate that the program is done:

```
RTX *;
PEND;
```

MI code example: MICRTPG2 complete program

In its consolidated state, this is the new MICRTPG2 program:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     program Name: MICRTPG2                                        */
/*                                                                   */
/*     programming Language: MI                                      */
/*                                                                   */
/*     Description: Initial version of MI program MICRTPG2,         */
/*                  which calls QPRCRTPG API.                        */
/*                                                                   */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
/* Entry point and associated parameters         */

ENTRY * (*ENTRY) EXT;
DCL SPCPTR FIL@ PARM;
DCL SPCPTR MBR@ PARM;
DCL OL *ENTRY (MBR@, FIL@) PARM EXT MIN(1);
DCL DD FIL CHAR(10) BAS(FIL@);
DCL DD MBR CHAR(10) BAS(MBR@);
DCL DD NUM_PARMS BIN( 4);

/* Control field for first time initialization   */

DCL DD READY CHAR( 1) INIT("0");

/* Binary offset into the space                  */

DCL DD BINOFFSET BIN(4) AUTO INIT(0);
DCL SPCPTR BINOFFSET@ AUTO INIT(BINOFFSET);

/* Pointers for accessing the space              */

DCL SPCPTR USRSPC;
DCL SYSPTR USRSPC@;

/* QCMDEXC and associated CL commands            */

DCL SYSPTR QCMDEXC INIT("QCMDEXC", CTX("QSYS"), TYPE(PGM));
DCL DD CLOVRCMD CHAR(65);
 DCL DD OVRSTR CHAR(39) DEF(CLOVRCMD) POS(1)
     INIT("OVRDBF MISRC 1234567890 MBR(1234567890)");
 DCL DD OVRSTR2 CHAR(26) DEF(CLOVRCMD) POS(40)
     INIT(" POSITION(*RRN 1234567890)");
 DCL DD FILNAM CHAR(10) DEF(CLOVRCMD) POS(14);
 DCL DD MBRNAM CHAR(10) DEF(CLOVRCMD) POS(29);
 DCL DD RECNUM ZND(10,0) DEF(CLOVRCMD) POS(55);
DCL SPCPTR CLOVRCMD@ INIT(CLOVRCMD);
DCL DD CLOVRLNG PKD(15,5) INIT(P'65');
DCL SPCPTR CLOVRLNG@ INIT(CLOVRLNG);
DCL OL QCMDOVROL (CLOVRCMD@, CLOVRLNG@) ARG;
DCL DD CLDLTCMD CHAR(12) INIT("DLTOVR MISRC");
DCL SPCPTR CLDLTCMD@ INIT(CLDLTCMD);
DCL DD CLDLTLNG PKD(15,5) INIT(P'12');
```

```
DCL SPCPTR CLDLTLNG@ INIT(CLDLTLNG);
DCL OL QCMDDLTOL (CLDLTCMD@, CLDLTLNG@) ARG;

/* CL06 and associated parameters               */

DCL SYSPTR CL06 INIT("CL06", TYPE(PGM));
DCL DD OFFSET PKD(15,5);
DCL SPCPTR OFFSET@ INIT(OFFSET);
DCL OL CL06OL (USRSPC, OFFSET@) ARG;

/* Access QTEMP address                                     */

DCL SYSPTR QTEMP@   BASPCO    POS(65);

/* Template for CRTS MI instruction             */

DCL DD CRTSTMPLT CHAR(160) BDRY(16);
 DCL DD TMPLTSPEC CHAR(8) DEF(CRTSTMPLT) POS(1);
  DCL DD TMPLTSIZE BIN(4) DEF(TMPLTSPEC) POS(1) INIT(160);
  DCL DD TMPLTBA   BIN(4) DEF(TMPLTSPEC) POS(5) INIT(0);
 DCL DD OBJID CHAR(32) DEF(CRTSTMPLT) POS(9);
  DCL DD SPCTYPE CHAR(1) DEF(OBJID) POS(1) INIT(X'19');
  DCL DD SPCSUBTYPE CHAR(1) DEF(OBJID) POS(2) INIT(X'EF');
  DCL DD SPCNAME CHAR(30) DEF(OBJID) POS(3) INIT("MICRTPG2");
 DCL DD OBJCRTOPT CHAR(4) DEF(CRTSTMPLT) POS(41) INIT(X'60020000');
 DCL DD OBJRCVOPTS CHAR(4) DEF(CRTSTMPLT) POS(45);
  DCL DD * CHAR(2) DEF(OBJRCVOPTS) POS(1) INIT(X'0000');
  DCL DD ASP CHAR(2) DEF(OBJRCVOPTS) POS(3) INIT(X'0000');
 DCL DD SPCSIZ BIN(4) DEF(CRTSTMPLT) POS(49) INIT(1);
 DCL DD INTSPCVAL CHAR(1) DEF(CRTSTMPLT) POS(53) INIT(X'00');
 DCL DD PERFCLASS CHAR(4) DEF(CRTSTMPLT) POS(54) INIT(X'00000000');
 DCL DD * CHAR(1) DEF(CRTSTMPLT) POS(58) INIT(X'00');
 DCL DD PUBAUT CHAR(2) DEF(CRTSTMPLT) POS(59) INIT(X'0000');
 DCL DD TMPLTEXTN BIN(4) DEF(CRTSTMPLT) POS(61) INIT(96);
 DCL SYSPTR CONTEXT DEF(CRTSTMPLT) POS(65);
 DCL SYSPTR ACCESSGRP DEF(CRTSTMPLT) POS(81);
 DCL SYSPTR USRPRF DEF(CRTSTMPLT) POS(97);
 DCL DD MAXSPCSIZ BIN(4) DEF(CRTSTMPLT) POS(113) INIT(0);
 DCL DD DOMAIN CHAR(2) DEF(CRTSTMPLT) POS(117) INIT(X'0001');
 DCL DD * CHAR(42) DEF(CRTSTMPLT) POS(119) INIT((42)X'00');
DCL SPCPTR CRTSTMPLT@ INIT(CRTSTMPLT);

/* QPRCRTPG and associated parameters           */

DCL DD PGM CHAR(20);
 DCL DD PGMNAM CHAR(10) DEF(PGM) POS(1);
 DCL DD PGMLIBNAM CHAR(10) DEF(PGM) POS(11) INIT("*CURLIB   ");
DCL SPCPTR PGM@ INIT(PGM);
DCL DD PGMTXT CHAR(50) INIT(" ");
DCL SPCPTR PGMTXT@ INIT(PGMTXT);
DCL DD PGMSRCF CHAR(20) INIT("*NONE");
DCL SPCPTR PGMSRCF@ INIT(PGMSRCF);
DCL DD PGMSRCM CHAR(10) INIT(" ");
DCL SPCPTR PGMSRCM@ INIT(PGMSRCM);
DCL DD PGMSRCCHG CHAR(13) INIT(" ");
DCL SPCPTR PGMSRCCHG@ INIT(PGMSRCCHG);
DCL DD PRTFNAM CHAR(20) INIT("QSYSPRT   *LIBL     ");
DCL SPCPTR PRTFNAM@ INIT(PRTFNAM);
DCL DD PRTSTRPAG BIN(4) INIT(1);
DCL SPCPTR PRTSTRPAG@ INIT(PRTSTRPAG);
DCL DD PGMPUBAUT CHAR(10) INIT("*ALL      ");
DCL SPCPTR PGMPUBAUT@ INIT(PGMPUBAUT);
DCL DD PGMOPTS(16) CHAR(11) INIT((1)"*LIST", *(2)(1)"*REPLACE",
        *(3)(1)"*XREF");
DCL SPCPTR PGMOPTS@ INIT(PGMOPTS);
DCL DD NUMOPTS BIN(4) INIT(3);
DCL SPCPTR NUMOPTS@ INIT(NUMOPTS);
```

```
DCL OL QPRCRTPGOL (USRSPC, BINOFFSET@, PGM@, PGMTXT@, PGMSRCF@,
                   PGMSRCM@, PGMSRCCHG@, PRTFNAM@, PRTSTRPAG@,
                   PGMPUBAUT@, PGMOPTS@, NUMOPTS@) ARG;
DCL SYSPTR QPRCRTPG INIT("QPRCRTPG", CTX("QSYS"), TYPE(PGM));

/* Start of instruction stream                  */

       STPLLEN NUM_PARMS;
       CMPNV(B) NUM_PARMS, 2 / EQ(PARM2);
       CPYBLAP FILNAM, 'MISRC', ' ';
       B PARM1;
PARM2: CPYBLA FILNAM, FIL;
PARM1: CPYBLA MBRNAM,MBR;
       CMPBLA(B) READY, '1' / EQ(SKIP);
       CPYBWP CONTEXT, QTEMP@;
       CRTS USRSPC@, CRTSTMPLT@;
       SETSPPFP USRSPC,USRSPC@;
       CPYBLA READY, '1';
SKIP:  CPYNV RECNUM, 1;
MORE:  CALLX QCMDEXC, QCMDOVROL, *;
       CPYNV OFFSET,1;
       CALLX CL06, CL06OL, *;
       SUBN(S) OFFSET, 1;
       ADDN(S) BINOFFSET, OFFSET;
       SETSPPO USRSPC, BINOFFSET;
       ADDN(S) RECNUM, 20;
       CALLX QCMDEXC, QCMDDLTOL, *;
       CMPNV(B) OFFSET, 1600 /EQ(MORE);
       CPYBLA PGMNAM, MBR;
       SETSPPO USRSPC, 0;
       CALLX QPRCRTPG, QPRCRTPGOL, *;
       RTX *;
       PEND;
```

Updated CL06 program

Following is the updated CL06 program:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: CL06                                            */
/*                                                                   */
/*     Programming Language: CL                                      */
/*                                                                   */
/*     Description: Load a source physical file member into the      */
/*                  *USRSPC addressed by &BUFFER.                    */
/*                                                                   */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
             PGM        PARM(&BUFFER &OFFSET)
             DCLF       FILE(MISRC)
             DCL        VAR(&BUFFER) TYPE(*CHAR) LEN(1600)
             DCL        VAR(&OFFSET) TYPE(*DEC) LEN(15 5)
LOOP:        RCVF
             MONMSG     MSGID(CPF0864 CPF4137) EXEC(GOTO CMDLBL(DONE))
             CHGVAR     VAR(%SST(&BUFFER &OFFSET 80)) VALUE(&SRCDTA)
             CHGVAR     VAR(&OFFSET) VALUE(&OFFSET + 80)
             IF         COND(&OFFSET *GT 1600) THEN(GOTO CMDLBL(DONE))
             GOTO       CMDLBL(LOOP)
DONE:        ENDPGM
```

Creating the MICRTPG2 program

This topic shows how to create the MICRTPG2 program and how to handle exceptions in the program.

To create the MICRTPG2 program, use:

```
DLTOVR MISRC
CALL CL03 (MISRC MICRTPG2)
```

After the successful creation of MICRTPG2, you can create any new MI programs by entering the following, where SourceFileName is an optional parameter:

```
CALL MICRTPG2 (MemberName SourceFileName)
```

Handling exceptions in the MICRTPG2 program

Some exceptions that are not being handled by the MICRTPG2 program might occur. For example, if you used MICRTPG2 to compile MICRTPG2 two times in succession, the exception MCH1401 occurs. This occurs because the most recent activation of the MICRTPG2 program has its own static storage and is not aware of the earlier instances of MICRTPG2 creating the space named MICRTPG2 in QTEMP.

To correct this problem, do the following:

1. Define an exception description that passes control to an internal exception handler:

   ```
   DCL EXCM DUPERROR EXCID(H'0E01') INT(M1401) IMD;
   ```

2. Define the internal entry point:

   ```
   ENTRY  M1401 INT;
   ```

3. Define related data elements for the M1401 exception:

   ```
   /* Exception description template for RETEXCPD   */

   DCL DD EXCPDBUF CHAR(200) BDRY(16);
    DCL DD BYTPRV BIN(4) DEF(EXCPDBUF) POS(1) INIT(200);
    DCL DD BYTAVL BIN(4) DEF(EXCPDBUF) POS(5);
    DCL DD EXCPID CHAR(2) DEF(EXCPDBUF) POS(9);
    DCL DD CMPLEN BIN(2) DEF(EXCPDBUF) POS(11);
    DCL DD CMPDTA CHAR(32) DEF(EXCPDBUF) POS(13);
    DCL DD MSGKEY CHAR(4) DEF(EXCPDBUF) POS(45);
    DCL DD EXCDTA CHAR(50) DEF(EXCPDBUF) POS(49);
     DCL SYSPTR EXC_OBJ@ DEF(EXCDTA) POS(1);
     DCL DD EXC_OBJ CHAR(32) DEF(EXCDTA) POS(17);
    DCL PTR INV_PTR DEF(EXCPDBUF) POS(97);
    DCL DD * CHAR(87) DEF(EXCPDBUF) POS(113);
   DCL SPCPTR EXCPDBUF@ INIT(EXCPDBUF);

   /* Template for RTNEXCP                         */

   DCL DD RTNTMPLT CHAR(19) BDRY(16);
    DCL PTR INV_PTR2 DEF(RTNTMPLT) POS(1);
    DCL DD * CHAR(1) DEF(RTNTMPLT) POS(17) INIT(X'00');
    DCL DD ACTION CHAR(2) DEF(RTNTMPLT) POS(18);
   DCL SPCPTR RTNTMPLT@ INIT(RTNTMPLT);
   ```

4. Retrieve the exception data associated with the MCH1401 exception:

   ```
   RETEXCPD EXCPDBUF@, X'01';
   ```

5. Compare the exception data object identifier to the space identifier you create. If they are the same, branch to label SAME:

   ```
   CMPBLA(B) EXC_OBJ, OBJID / EQ(SAME);
   ```

   a. If the exception data object identifier and the space identifier are not the same, the program is truly in an unexpected error condition and the exception description needs to be disabled:

      ```
      MODEXCPD DUPERROR, X'2000', X'01';
      ```

Retry the failing instruction. As the exception description is disabled, the exception is sent to the caller of the program:

```
        CPYBLA ACTION, X'0000';
        B E1401;
```

b. If the exception data object identifier and the space identifier are the same, the static storage must have been effectively reset. The program reassigns USRSPC@ by using the returned system pointer in the exception data and continues with the next instruction following the failed CRTS:

```
SAME:  CPYBWP USRSPC@, EXC_OBJ@;
       CPYBLA ACTION, X'0100';
E1401: CPYBWP INV_PTR2, INV_PTR;
       RTNEXCP RTNTMPLT@;
       PEND;
```

MI code example: MICRTPG2 complete program (enhanced)

In its consolidated state, this is the new MICRTPG2 program:

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     program Name: MICRTPG2                                        */
/*                                                                   */
/*     programming Language: MI                                      */
/*                                                                   */
/*     Description: Enhanced version of MI program MICRTPG2,         */
/*                  which provides for exception handling.           */
/*                                                                   */
/*                                                                   */
/*     Header Files Included: None                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
/* Entry point and associated parameters         */

ENTRY * (*ENTRY) EXT;
DCL SPCPTR FIL@ PARM;
DCL SPCPTR MBR@ PARM;
DCL OL *ENTRY (MBR@, FIL@) PARM EXT MIN(1);
DCL DD FIL CHAR(10) BAS(FIL@);
DCL DD MBR CHAR(10) BAS(MBR@);
DCL DD NUM_PARMS BIN( 4);

/* Control field for first time initialization   */

DCL DD READY CHAR( 1) INIT("0");

/* Binary offset into the space                  */

DCL DD BINOFFSET BIN(4) AUTO INIT(0);
DCL SPCPTR BINOFFSET@ AUTO INIT(BINOFFSET);

/* Pointers for accessing the space              */

DCL SPCPTR USRSPC;
DCL SYSPTR USRSPC@;

/* QCMDEXC and associated CL commands            */

DCL SYSPTR QCMDEXC INIT("QCMDEXC", CTX("QSYS"), TYPE(PGM));
DCL DD CLOVRCMD CHAR(65);
 DCL DD OVRSTR CHAR(39) DEF(CLOVRCMD) POS(1)
```

```
     INIT("OVRDBF MISRC 1234567890 MBR(1234567890)");
 DCL DD OVRSTR2 CHAR(26) DEF(CLOVRCMD) POS(40)
     INIT(" POSITION(*RRN 1234567890)");
 DCL DD FILNAM CHAR(10) DEF(CLOVRCMD) POS(14);
 DCL DD MBRNAM CHAR(10) DEF(CLOVRCMD) POS(29);
 DCL DD RECNUM ZND(10,0) DEF(CLOVRCMD) POS(55);
DCL SPCPTR CLOVRCMD@ INIT(CLOVRCMD);
DCL DD CLOVRLNG PKD(15,5) INIT(P'65');
DCL SPCPTR CLOVRLNG@ INIT(CLOVRLNG);
DCL OL QCMDOVROL (CLOVRCMD@, CLOVRLNG@) ARG;
DCL DD CLDLTCMD CHAR(12) INIT("DLTOVR MISRC");
DCL SPCPTR CLDLTCMD@ INIT(CLDLTCMD);
DCL DD CLDLTLNG PKD(15,5) INIT(P'12');
DCL SPCPTR CLDLTLNG@ INIT(CLDLTLNG);
DCL OL QCMDDLTOL (CLDLTCMD@, CLDLTLNG@) ARG;

/* CL06 and associated parameters               */

DCL SYSPTR CL06 INIT("CL06", TYPE(PGM));
DCL DD OFFSET PKD(15,5);
DCL SPCPTR OFFSET@ INIT(OFFSET);
DCL OL CL06OL (USRSPC, OFFSET@) ARG;

/* Access QTEMP address                                         */

DCL SYSPTR QTEMP@   BASPCO    POS(65);

/* Template for CRTS MI instruction             */

DCL DD CRTSTMPLT CHAR(160) BDRY(16);
 DCL DD TMPLTSPEC CHAR(8) DEF(CRTSTMPLT) POS(1);
  DCL DD TMPLTSIZE BIN(4) DEF(TMPLTSPEC) POS(1) INIT(160);
  DCL DD TMPLTBA   BIN(4) DEF(TMPLTSPEC) POS(5) INIT(0);
 DCL DD OBJID CHAR(32) DEF(CRTSTMPLT) POS(9);
  DCL DD SPCTYPE CHAR(1) DEF(OBJID) POS(1) INIT(X'19');
  DCL DD SPCSUBTYPE CHAR(1) DEF(OBJID) POS(2) INIT(X'EF');
  DCL DD SPCNAME CHAR(30) DEF(OBJID) POS(3) INIT("MICRTPG2");
 DCL DD OBJCRTOPT CHAR(4) DEF(CRTSTMPLT) POS(41) INIT(X'60020000');
 DCL DD OBJRCVOPTS CHAR(4) DEF(CRTSTMPLT) POS(45);
  DCL DD * CHAR(2) DEF(OBJRCVOPTS) POS(1) INIT(X'0000');
  DCL DD ASP CHAR(2) DEF(OBJRCVOPTS) POS(3) INIT(X'0000');
 DCL DD SPCSIZ BIN(4) DEF(CRTSTMPLT) POS(49) INIT(1);
 DCL DD INTSPCVAL CHAR(1) DEF(CRTSTMPLT) POS(53) INIT(X'00');
 DCL DD PERFCLASS CHAR(4) DEF(CRTSTMPLT) POS(54) INIT(X'00000000');
 DCL DD * CHAR(1) DEF(CRTSTMPLT) POS(58) INIT(X'00');
 DCL DD PUBAUT CHAR(2) DEF(CRTSTMPLT) POS(59) INIT(X'0000');
 DCL DD TMPLTEXTN BIN(4) DEF(CRTSTMPLT) POS(61) INIT(96);
 DCL SYSPTR CONTEXT DEF(CRTSTMPLT) POS(65);
 DCL SYSPTR ACCESSGRP DEF(CRTSTMPLT) POS(81);
 DCL SYSPTR USRPRF DEF(CRTSTMPLT) POS(97);
 DCL DD MAXSPCSIZ BIN(4) DEF(CRTSTMPLT) POS(113) INIT(0);
 DCL DD DOMAIN CHAR(2) DEF(CRTSTMPLT) POS(117) INIT(X'0001');
 DCL DD * CHAR(42) DEF(CRTSTMPLT) POS(119) INIT((42)X'00');
DCL SPCPTR CRTSTMPLT@ INIT(CRTSTMPLT);

/* QPRCRTPG and associated parameters           */

DCL DD PGM CHAR(20);
 DCL DD PGMNAM CHAR(10) DEF(PGM) POS(1);
 DCL DD PGMLIBNAM CHAR(10) DEF(PGM) POS(11) INIT("*CURLIB   ");
DCL SPCPTR PGM@ INIT(PGM);
DCL DD PGMTXT CHAR(50) INIT(" ");
DCL SPCPTR PGMTXT@ INIT(PGMTXT);
DCL DD PGMSRCF CHAR(20) INIT("*NONE");
DCL SPCPTR PGMSRCF@ INIT(PGMSRCF);
DCL DD PGMSRCM CHAR(10) INIT(" ");
DCL SPCPTR PGMSRCM@ INIT(PGMSRCM);
```

```
DCL DD PGMSRCCHG CHAR(13) INIT(" ");
DCL SPCPTR PGMSRCCHG@ INIT(PGMSRCCHG);
DCL DD PRTFNAM CHAR(20) INIT("QSYSPRT   *LIBL     ");
DCL SPCPTR PRTFNAM@ INIT(PRTFNAM);
DCL DD PRTSTRPAG BIN(4) INIT(1);
DCL SPCPTR PRTSTRPAG@ INIT(PRTSTRPAG);
DCL DD PGMPUBAUT CHAR(10) INIT("*ALL      ");
DCL SPCPTR PGMPUBAUT@ INIT(PGMPUBAUT);
DCL DD PGMOPTS(16) CHAR(11) INIT((1)"*LIST", *(2)(1)"*REPLACE",
       *(3)(1)"*XREF");
DCL SPCPTR PGMOPTS@ INIT(PGMOPTS);
DCL DD NUMOPTS BIN(4) INIT(3);
DCL SPCPTR NUMOPTS@ INIT(NUMOPTS);
DCL OL QPRCRTPGOL (USRSPC, BINOFFSET@, PGM@, PGMTXT@, PGMSRCF@,
                   PGMSRCM@, PGMSRCCHG@, PRTFNAM@, PRTSTRPAG@,
                   PGMPUBAUT@, PGMOPTS@, NUMOPTS@) ARG;
DCL SYSPTR QPRCRTPG INIT("QPRCRTPG", CTX("QSYS"), TYPE(PGM));

/* Exception Description Monitor for MCH1401     */

DCL EXCM DUPERROR EXCID(H'0E01') INT(M1401) IMD;

/* Start of instruction stream                   */

       STPLLEN NUM_PARMS;
       CMPNV(B) NUM_PARMS, 2 / EQ(PARM2);
       CPYBLAP FILNAM, 'MISRC', ' ';
       B PARM1;
PARM2: CPYBLA FILNAM, FIL;
PARM1: CPYBLA MBRNAM,MBR;
       CMPBLA(B) READY, '1' / EQ(SKIP);
       CPYBWP CONTEXT, QTEMP@;
       CRTS USRSPC@, CRTSTMPLT@;
       SETSPPFP USRSPC,USRSPC@;
       CPYBLA READY, '1';
SKIP:  CPYNV RECNUM, 1;
MORE:  CALLX QCMDEXC, QCMDOVROL, *;
       CPYNV OFFSET,1;
       CALLX CL06, CL06OL, *;
       SUBN(S) OFFSET, 1;
       ADDN(S) BINOFFSET, OFFSET;
       SETSPPO USRSPC, BINOFFSET;
       ADDN(S) RECNUM, 20;
       CALLX QCMDEXC, QCMDDLTOL, *;
       CMPNV(B) OFFSET, 1600 /EQ(MORE);
       CPYBLA PGMNAM, MBR;
       SETSPPO USRSPC, 0;
       CALLX QPRCRTPG, QPRCRTPGOL, *;
       RTX *;

/* Entry point for internal exception handler    */

ENTRY  M1401 INT;

/* Exception description template for RETEXCPD   */

DCL DD EXCPDBUF CHAR(200) BDRY(16);
 DCL DD BYTPRV BIN(4) DEF(EXCPDBUF) POS(1) INIT(200);
 DCL DD BYTAVL BIN(4) DEF(EXCPDBUF) POS(5);
 DCL DD EXCPID CHAR(2) DEF(EXCPDBUF) POS(9);
 DCL DD CMPLEN BIN(2) DEF(EXCPDBUF) POS(11);
 DCL DD CMPDTA CHAR(32) DEF(EXCPDBUF) POS(13);
 DCL DD MSGKEY CHAR(4) DEF(EXCPDBUF) POS(45);
 DCL DD EXCDTA CHAR(50) DEF(EXCPDBUF) POS(49);
  DCL SYSPTR EXC_OBJ@ DEF(EXCDTA) POS(1);
  DCL DD EXC_OBJ CHAR(32) DEF(EXCDTA) POS(17);
 DCL PTR INV_PTR DEF(EXCPDBUF) POS(97);
```

```
 DCL DD * CHAR(87) DCF(EXCPDBUF) POS(113);
DCL SPCPTR EXCPDBUF@ INIT(EXCPDBUF);

/* Template for RTNEXCP                          */

DCL DD RTNTMPLT CHAR(19) BDRY(16);
 DCL PTR INV_PTR2 DEF(RTNTMPLT) POS(1);
 DCL DD * CHAR(1) DEF(RTNTMPLT) POS(17) INIT(X'00');
 DCL DD ACTION CHAR(2) DEF(RTNTMPLT) POS(18);
DCL SPCPTR RTNTMPLT@ INIT(RTNTMPLT);

/* Start of internal handler                     */

       RETEXCPD EXCPDBUF@, X'01';
       CMPBLA(B) EXC_OBJ, OBJID / EQ(SAME);
       MODEXCPD DUPERROR, X'2000', X'01';
       CPYBLA ACTION, X'0000';
       B E1401;
SAME:  CPYBWP USRSPC@, EXC_OBJ@;
       CPYBLA ACTION, X'0100';
E1401: CPYBWP INV_PTR2, INV_PTR;
       RTNEXCP RTNTMPLT@;
       PEND;
```

Related tasks:

"Enhanced version of the MICRTPG program" on page 513
The enhanced version of the MICRTPG program (named MICRTPG2) incorporates the functions of the CL03 program and the CL05 program.

Related reference:

"Compiling an MI program" on page 500
If you enter the source into a source physical file, you can now compile the source and create an MI program. To create the program, use the Create Program (QPRCRTPG) API. Test and debug the program if it has errors. You can also declare an exception handler for the program.

Example: Common MI programming techniques

This example MI program demonstrates additional programming techniques. For each object found in the QTEMP library, the program sends a message to the interactive user message queue showing the object name, type, and subtype.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     Program Name: MISC1                                           */
/*                                                                   */
/*     Programming Language: MI                                      */
/*                                                                   */
/*     Description: This program materializes the objects found      */
/*                  within the QTEMP library (context).  For each    */
/*                  object found, a message is sent to the           */
/*                  interactive user message queue showing the       */
/*                  name of the object and the object's type and     */
/*                  subtype.                                         */
/*                                                                   */
/*                  Several new MI instructions are used by this     */
/*                  program:                                         */
/*                                                                   */
/*                  1. Materialize Context (MATCTX)                  */
/*                  2. Modify Automatic Storage (MODASA)             */
/*                  3. Divide (DIV)                                  */
/*                  4. Convert Hex to Character (CVTHC)              */
/*                  5. Override Program Attributes (OVRPGATR)        */
```

```
/*                                                                  */
/*                                                                  */
/*     Header Files Included: None                                  */
/*                                                                  */
/*                                                                  */
/********************************************************************/
/* Entry point                                                      */

ENTRY * EXT;

/* Declare layout of Process Communications Object (PCO)            */

/* The PCO is a control area that is unique to each job on the      */
/* system.  Within the PCO, there are two data elements that can    */
/* be used.  The first is a space pointer to the system entry       */
/* point table (SEPT), the second is the address of the QTEMP       */
/* library.  The use of any other data element in the PCO is NOT    */
/* supported.                                                       */

DCL DD PCO CHAR(80) BASPCO;
 DCL SPCPTR SEPT@             DEF(PCO)        POS( 1);
 DCL SYSPTR QTEMP@            DEF(PCO)        POS(65);

/* The SEPT is an array of system pointers that address IBM         */
/* programs in QSYS.  Within this array of pointers, some of the    */
/* offsets represent fixed (upward compatible) assignments.  All    */
/* IBM i APIs, for instance, are fixed at certain offsets within    */
/* the SEPT and you can call these APIs directly via the SEPT.      */
/* Calling APIs in this way avoids having to resolve to the API     */
/* (that is, performance is improved) and prevents someone from     */
/* placing their version of the API earlier in the library list     */
/* than the IBM-supplied API (that is, avoids counterfeits).        */
/* All APIs, and their offsets, can be found in the source member   */
/* QLIEPTI of file H in the optionally installed QSYSINC library.   */
/* You should only use the SEPT for those programs identified in    */
/* member QLIEPTI.  The use of any other SEPT offsets is NOT        */
/* supported.                                                       */

/* Because the offset values in member QLIEPTI are oriented to the */
/* C language, they are assuming a base of 0.  Because MI arrays    */
/* use a default base of 1, we will declare the SEPT array with     */
/* an explicit base of 0.  Because the array can grow over time     */
/* (and we don't necessarily want to have to change the upper       */
/* bound every release), we'll just define the array as having 2    */
/* elements and use the OVRPGATR instruction later in the program   */
/* to instruct the translator to ignore the array bounds when       */
/* referring to the array.  For example, later we will use          */
/* SEPT(4267) to call the Send Nonprogram Message (QMHSNDM) API.    */

DCL SYSPTR SEPT(0:1) BAS(SEPT@);  /* use Base 0 to match QLIEPTI    */

/* Declare template for Materialize Context (MATCTX)                */

DCL DD MATCTXOPTS CHAR(44);
 DCL DD MATCTXCTL    CHAR( 2) DEF(MATCTXOPTS) POS( 1) INIT(X'0500');
 DCL DD MATCTXSELCTL CHAR(42) DEF(MATCTXOPTS) POS( 3);

/* Declare Small Receiver for initial MATCTX                        */

DCL DD S_RECEIVER CHAR(8) BDRY(16);
 DCL DD S_BYTPRV     BIN( 4)  DEF(S_RECEIVER) POS( 1) INIT(8);
 DCL DD S_BYTAVL     BIN( 4)  DEF(S_RECEIVER) POS( 5);
DCL SPCPTR S_RECEIVER@ INIT(S_RECEIVER);

/* Declare Large Receiver Layout for second MATCTX                  */

DCL DD L_RECEIVER    CHAR(129) BAS(L_RECEIVER@);
```

```
 DCL DD L_BYTPRV     BIN( 4)  DEF(L_RECEIVER) POS( 1);
 DCL DD L_BYTAVL     BIN( 4)  DEF(L_RECEIVER) POS( 5);
 DCL DD L_CONTEXT    CHAR(32) DEF(L_RECEIVER) POS( 9);
  DCL DD L_OBJ_TYPE  CHAR( 1) DEF(L_CONTEXT)  POS( 1);
  DCL DD L_OBJ_STYPE CHAR( 1) DEF(L_CONTEXT)  POS( 2);
  DCL DD L_OBJ_NAME  CHAR(30) DEF(L_CONTEXT)  POS( 3);
 DCL DD L_CTX_OPTS   CHAR( 4) DEF(L_RECEIVER) POS(41);
 DCL DD L_RCV_OPTS   CHAR( 4) DEF(L_RECEIVER) POS(45);
 DCL DD L_SPC_SIZ    BIN( 4)  DEF(L_RECEIVER) POS(49);
 DCL DD L_SPC_IVAL   CHAR( 1) DEF(L_RECEIVER) POS(53);
 DCL DD L_PERF_CLS   CHAR( 4) DEF(L_RECEIVER) POS(54);
 DCL DD *            CHAR( 7) DEF(L_RECEIVER) POS(58);
 DCL DD *            CHAR(16) DEF(L_RECEIVER) POS(65);
 DCL SYSPTR L_ACC_GROUP;
 DCL DD L_EXT_ATTR   CHAR( 1) DEF(L_RECEIVER) POS(81);
 DCL DD *            CHAR( 7) DEF(L_RECEIVER) POS(82);
 DCL DD L_TIMESTAMP  CHAR( 8) DEF(L_RECEIVER) POS(89);
 DCL DD L_ENTRY      CHAR(32) DEF(L_RECEIVER) POS(97);

/* Individual object entry layout                                   */

DCL DD OBJ_ENTRY     CHAR(32) BAS(OBJ_ENTRY@);
 DCL DD OBJ_INFO_X   CHAR( 2) DEF(OBJ_ENTRY)  POS( 1);
  DCL DD OBJ_TYPE_X  CHAR( 1) DEF(OBJ_INFO_X) POS( 1);
  DCL DD OBJ_STYPE_X CHAR( 1) DEF(OBJ_INFO_X) POS( 2);
 DCL DD OBJ_NAME     CHAR(30) DEF(OBJ_ENTRY)  POS( 3);

/* Define basing pointers:                                          */

DCL SPCPTR L_RECEIVER@;
DCL SPCPTR OBJ_ENTRY@;

/* Define various working variables                                 */

DCL DD SIZE          BIN( 4);    /* number of objects materialized */
DCL DD NUM_DONE      BIN( 4)     /* number of objects processed    */
                       AUTO INIT(0);
/* Define needed parameters for QMHSNDM API                         */

DCL DD MSG_ID        CHAR (7) INIT("       ");
DCL SPCPTR MSG_ID@   INIT(MSG_ID);
DCL DD MSG_FILE      CHAR(20) INIT("                    ");
DCL SPCPTR MSG_FILE@ INIT(MSG_FILE);
DCL DD MSG_TEXT      CHAR(57);
 DCL DD *            CHAR( 8) DEF(MSG_TEXT)   POS( 1)
                       INIT("OBJECT: ");
 DCL DD OBJ_NAME_T   CHAR(30) DEF(MSG_TEXT)   POS( 9);
 DCL DD *            CHAR(15) DEF(MSG_TEXT)   POS(39)
                       INIT(" TYPE/SUBTYPE: ");
 DCL DD OBJ_INFO_C   CHAR( 4) DEF(MSG_TEXT)   POS(54);
  DCL DD OBJ_TYPE_C  CHAR( 2) DEF(OBJ_INFO_C) POS( 1);
  DCL DD OBJ_STYPE_C CHAR( 2) DEF(OBJ_INFO_C) POS( 3);
DCL SPCPTR MSG_TEXT@ INIT(MSG_TEXT);
DCL DD MSG_SIZE      BIN( 4)  INIT(57);
DCL SPCPTR MSG_SIZE@ INIT(MSG_SIZE);
DCL DD MSG_TYPE      CHAR(10) INIT("*INFO     ");
DCL SPCPTR MSG_TYPE@ INIT(MSG_TYPE);
DCL DD MSG_QS        CHAR(20) INIT("*REQUESTER          ");
DCL SPCPTR MSG_QS@   INIT(MSG_QS);
DCL DD MSG_QSN       BIN( 4)  INIT(1);
DCL SPCPTR MSG_QSN@  INIT(MSG_QSN);
DCL DD REPLY_Q       CHAR(20) INIT("                    ");
DCL SPCPTR REPLY_Q@  INIT(REPLY_Q);
DCL DD MSG_KEY       CHAR( 4);
DCL SPCPTR MSG_KEY@  INIT(MSG_KEY);
DCL DD ERR_COD       BIN( 4)  INIT(0);
DCL SPCPTR ERR_COD@  INIT(ERR_COD);
```

```
DCL OL QMHSNDMOL (MSG_ID@, MSG_FILE@, MSG_TEXT@, MSG_SIZE@,
                  MSG_TYPE@, MSG_QS@, MSG_QSN@, REPLY_Q@,
                  MSG_KEY@, ERR_COD@) ARG;

/* Start the instruction stream                                    */

/* Materialize the amount of storage needed to store object info   */

            MATCTX     S_RECEIVER@, QTEMP@, MATCTXOPTS;

/* If no objects are in the library, then exit                     */

            CMPNV(B)   S_BYTAVL, 96 / EQ(DONE);

/* Allocate the necessary storage (we could also have used CRTS
   to allocate the storage and a SPCPTR to the space for the
   large receiver variable)                                         */

            MODASA     L_RECEIVER@, S_BYTAVL;

/* Set the bytes provided field to indicate the allocated storage  */

            CPYNV      L_BYTPRV, S_BYTAVL;

/* Materialize the objects within the library                      */

            MATCTX     L_RECEIVER@, QTEMP@, MATCTXOPTS;

/* Calculate how many objects were returned:                       */
/* 1. Find the lower of bytes provided and bytes available         */
/*    (L_BYTPRV and L_BYTAVL) as the number of objects could have  */
/*    changed since the first materialize                          */
/* 2. Subtract the size of the fixed MATCTX header (96)            */
/* 3. Divide the remainder by the size of each entry returned      */

            CMPNV(B)   L_BYTPRV, L_BYTAVL / HI(ITS_AVL);
            CPYNV      SIZE, L_BYTPRV;
            B          CONTINUE;
ITS_AVL:    CPYNV      SIZE, L_BYTAVL;
CONTINUE:   SUBN(SB)   SIZE, 96 / ZER(DONE);
            DIV        SIZE, SIZE, 32;

/* Address the first object returned                               */

            SETSPP     OBJ_ENTRY@, L_ENTRY;

/* Loop through all materialized entries                           */

MORE:

/* Convert the hex object type and subtype to character form       */

            CVTHC      OBJ_INFO_C, OBJ_INFO_X;

/* Copy the object name to the message variable                    */

            CPYBLA     OBJ_NAME_T, OBJ_NAME;

/* Unconstrain the array bounds (at compile time)                  */

            OVRPGATR   1,3;

/* Send a message to caller's msg queue containing the object info */

            CALLX      SEPT(4267), QMHSNDMOL, *;

/* resume normal array constraint                                  */
```

```
            OVRPGATR   1,4;

/* and move on to the next entry                                    */

            ADDN(S)    NUM_DONE, 1;
            ADDSPP     OBJ_ENTRY@, OBJ_ENTRY@, 32;
            CMPNV(B)   NUM_DONE, SIZE / LO(MORE);

/* When all entries are processed, end the program.                 */
/*                                                                  */
/* Note that this program may not actually display all objects      */
/* in QTEMP.  If L_BYTAVL is greater than L_BYTPRV, additional      */
/* objects were inserted into QTEMP between the time of the         */
/* "small" MATCTX and the "large" MATCTX.  The processing of these */
/* additional objects is not addressed in this program and is       */
/* the responsibility of the user of this program.                  */
/*                                                                  */

DONE:       RTX        *;
            PEND;
```

Program storage

Program activation and program invocation are needed to run an MI program.

Program activation is the process of allocating and initializing static storage for the program. *Program invocation* is the process of allocating and initializing automatic storage.

Program activation and static storage

Program activation can be done explicitly through the Activate Program (ACTPG) instruction or implicitly by using a call external (CALLX) instruction when the called program has not been previously activated. Program activation typically occurs only once within a job or process. Program activation is not reset by an RTX instruction within the called program (the program is still considered to be in an activated state). This means that all static storage on subsequent calls (CALLXs) to the program are found in a last-used state, not in a reinitialized state. If a programmer wants to reinitialize the static storage associated with a program activation, this can be accomplished through the deactivate program (DEACTPG) instruction so that the next call (CALLX or ACTPG) causes a new activation of the program.

Program invocation and automatic storage

Program invocation, on the other hand, occurs every time a program is called with a CALLX instruction. Automatic storage is reinitialized if a discrete INIT value was specified on the declare (DCL) statement. (If the INIT was allowed to be the default, then whether initialization occurs for the field is determined by an option of the QPRCRTPG API when the program was created.) If you have not already done so, review all of the option template values available on the QPRCRTPG API before developing your MI applications.

Common API programming errors

This topic contains information about common API programming errors and provides correct and incorrect program examples.

Note: Do not assume that an API can do things other than what the API documentation states. If the API documentation does not state specifically that something is allowed, it probably is not.

Using the error code parameter

The error code parameter provides a way for you to determine whether an API encounters any errors. Here are the program examples that show the incorrect and correct ways of using the error code parameter.

The examples in this topic present a program that is used for creating a user space.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Incorrect program example: Using the error code parameter

The common error shown in this example is the use of the error code structure to indicate to the API not to send exception messages for errors found. Additionally, the example does not examine the error code structure to determine if the API call was successful or not. To demonstrate the improper use of the error code structure, an incorrect value is used on the replace parameter of the QUSCRTUS API. The replace parameter is a required parameter. The coded error (*XXXXXXX) is shown at location (1) in the incorrect and also at location (2) in the correct coding.

Both the incorrect (3) and correct coding (4) show the program monitoring for any error from the call to the API. However, the program does not examine the bytes available field after calling the QUSCRTUS API.

Because of the error on the replace parameter, the requested user space is not created. The calling program, however, is not aware of this as shown at (5).

```
 ****************************************************************
 *
 *Program Name: PGM1
 *
 *Program Language:  RPG
 *
 *Description: This sample program illustrates the incorrect
 *             way of using the error code parameter.
 *
 *Header Files Included: QUSEC - Error Code Parameter
 *
 *APIs Used:  QUSCRTUS - Create User Space
 *
 ****************************************************************
 * BRING IN THE ERROR STRUCTURE FROM QSYSINC
I/COPY QSYSINC/QRPGSRC,QUSEC
 **
ISPCNAM      DS
I I            'SPCNAME   '              1  10 SPC
I I            'PAM       '             11  20 LIB
 ** OTHER ASSORTED VARIABLES
I            DS
I I            2000                  B   1   40SIZ
I                                    B   5   80START
I I            X'00'                     9   9 INTVAL
 *
 * Initialize the bytes provided field (QUSBNDB) of the error code
 * structure.  Languages such as RPG and CL tend to initialize the bytes
 * provided field to blanks, which when passed to an API is viewed as a
 * very large (and incorrect) binary value.  If you receive CPF3CF1 when
 * calling an API, the bytes provided field should be the first field
 * you examine as part of problem determination.
C                     Z-ADD16        QUSBNB               (3)
 *
 * CREATE THE SPACE TO HOLD THE DATA
C                     CALL 'QUSCRTUS'
```

```
C                     PARM           SPCNAM
C                     PARM 'EXT_ATTR'EXTATR 10
C                     PARM           SIZ
C                     PARM           INTVAL
C                     PARM '*ALL    'PUBAUT 10
C                     PARM 'NO TEXT 'TXTDSC 50
C                     PARM '*XXXXXXX'REPLAC 10            (1)
C                     PARM           QUSBN
 ** Program does not check the error code parameter       (5)
 **
C                     SETON                     LR
```

Correct program example: Using the error code parameter

You can add code to help you discover what errors might be in a program. In the following example program, code has been added to monitor error information that is passed back in the error code parameter (QUSBN). The code at (6) has been added to check the error code parameter for any messages and to display the exception identifier to the user if any errors are found. The incorrectly coded program does no checking for the error code parameter, as shown at (5).

```
 ****************************************************************
 *
 *Program Name: PGM2
 *
 *Program Language:  RPG
 *
 *Description: This sample program illustrates the correct
 *             way of using the error code parameter.
 *
 *Header Files Included: QUSEC - Error Code Parameter
 *
 *APIs Used:  QUSCRTUS - Create User Space
 *
 ****************************************************************
 * BRING IN THE ERROR STRUCTURE FROM QSYSINC
I/COPY QSYSINC/QRPGSRC,QUSEC
 **
ISPCNAM      DS
I I            'SPCNAME   '              1  10 SPC
I I            'QTEMP     '             11  20 LIB
 ** OTHER ASSORTED VARIABLES
I            DS
I I            2000                  B   1   40SIZ
I                                    B   5   80START
I I            X'00'                     9   9 INTVAL
 *
C                     Z-ADD16        QUSBNB              (4)
 *
 * CREATE THE SPACE TO HOLD THE DATA
C                     CALL 'QUSCRTUS'
C                     PARM           SPCNAM
C                     PARM 'EXT_ATTR'EXTATR 10
C                     PARM           SIZ
C                     PARM           INTVAL
C                     PARM '*ALL    'PUBAUT 10
C                     PARM 'NO TEXT 'TXTDSC 50
C                     PARM '*XXXXXXX'REPLAC 10            (2)
C                     PARM           QUSBN
 **
 * DISPLAY EXCEPTION IDENTIFIER TO THE USER
C           QUSBNC    IFGT *ZEROS                         (6)
C                     EXSR DSPERR
C                     END
 *
C                     SETON                     LR
```

```
 *
C             DSPERR  BEGSR
C                     DSPLY          QUSBND
C                     ENDSR
```

Defining data structures

When a data structure is defined for use with an API, the structure must be built to receive what the API returns. Here are the program examples that show the incorrect and correct ways of defining data structures. You can prevent errors by using IBM-supplied data structures rather than creating your own data structures.

For information about IBM-supplied data structures that are contained in the QSYSINC library, see "Include files and the QSYSINC library" on page 59.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Incorrect program example: Defining a data structure

When the program that defines a data structure is run, it does the following:

- Creates a user space
- Retrieves a list of active jobs
- Displays the first part of a job name
- Deletes the user space that held the data

In this example, the data structure to be used with the QUSLJOB API has been defined incorrectly. The incorrectly defined variables are JNAME and USRNAM. The JNAME length is defined as 1 through 12 and the USRNAM length as 13 through 20. This is shown at (1). The data displayed (JNAME variable) will be incorrect. The correct coding is shown at (2).

```
 *****************************************************************
 *
 *Program Name: PGM1
 *
 *Program Language:  RPG
 *
 *Description: This sample program illustrates the incorrect
 *             way of defining data structures.
 *
 *Header Files Included: QUSEC - Error Code Parameter
 *                       QUSGEN - User Space Format for Generic Header
 *
 *APIs Used:  QUSCRTUS  - Create User Space
 *            QUSLJOB   - List Job
 *            QUSRTVUS  - Retrieve User Space
 *            QUSDLTUS  - Delete User Space
 *****************************************************************
 * THIS PROGRAM WILL CREATE THE NECESSARY SPACE AND THEN CALL
 * THE QUSLJOB API TO GET A LIST OF ALL ACTIVE JOBS ON THE SYSTEM.
 * THE FIRST JOB NAME/USER WILL BE DISPLAYED TO THE USER.
 *
 * BRING IN THE USER SPACE GENERIC HEADER
I/COPY QSYSINC/QRPGSRC,QUSGEN
 * BRING IN THE ERROR STRUCTURE FROM QSYSINC
I/COPY QSYSINC/QRPGSRC,QUSEC
 ** JOB NAME STRUCTURE FOR CALLING QUSLJOB
IJOBNAM     DS
I I            '*ALL      '              1  10 JOB
I I            '*ALL      '             11  20 USER
I I            '*ALL  '                 21  26 JOBNUM
 ** JOBL0100 FORMAT RETURNED FROM QUSLJOB API
```

```
 ** INCORRECTLY CODE THE JNAME/USRNAM LENGTHS
IRECVR     DS
I                                        1  12 JNAME        (1)
I                                       13  20 USRNAM       (1)
I                                       21  26 JOBNBR
I                                       27  42 JOBID
I                                       43  52 JSTAT
I                                       53  53 JTYPE
I                                       54  54 JSUBT
I                                       55  56 RESRV
 **
ISPCNAM      DS
I I            'SPCNAME   '             1  10 SPC
I I            'QTEMP     '            11  20 LIB
** OTHER ASSORTED VARIABLES
I           DS
I I            2000                  B   1   40SIZ
I I                                  B   5   80START
I I                                  B   9  120LENDTA
I I            X'00'                    13   13INTVAL
 *
 * SET UP TO ACCEPT EXCEPTIONS
C                     Z-ADD*ZEROS    QUSBNB
 *
 * CREATE THE SPACE TO HOLD THE DATA
C                     CALL 'QUSCRTUS'
C                     PARM           SPCNAM
C                     PARM 'EXT_ATTR'EXTATR 10
C                     PARM           SIZ
C                     PARM           INTVAL
C                     PARM '*ALL    'PUBAUT 10
C                     PARM 'TEXT DSC'TXTDSC 50
C                     PARM '*YES    'REPLAC 10
C                     PARM           QUSBN
 *
 * CALL THE API TO LIST THE ACTIVE JOBS
C                     CALL 'QUSLJOB'
C                     PARM           SPCNAM
C                     PARM 'JOBL0100'FORMAT  8
C                     PARM           JOBNAM
C                     PARM '*ACTIVE 'STAT   10
C                     PARM           QUSBN
 *
 * RETRIEVE THE OFFSET OF THE FIRST LIST ENTRY FROM THE SPACE
C                     Z-ADD1         START
C                     Z-ADD140       LENDTA
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM
C                     PARM           START
C                     PARM           LENDTA
C                     PARM           QUSBP
C                     PARM           QUSBN
 *
 * RETRIEVE THE FIRST LIST ENTRY
C           QUSBPQ    ADD  1         START
C                     Z-ADD56        LENDTA
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM
C                     PARM           START
C                     PARM           LENDTA
C                     PARM           RECVR
C                     PARM           QUSBN
 *
 * DISPLAY THE JOB NAME
C                     DSPLY          JNAME
 ****************************
 * When displayed, JNAME    *
```

```
 * will look something like *
 * 'QCPF       QS'          *
 ****************************
 * DELETE THE SPACE THAT HELD THE DATA
C                     CALL 'QUSDLTUS'
C                     PARM           SPCNAM
C                     PARM           QUSBN
 **
C                     SETON                     LR
```

Correct program example: Defining a data structure

The following program uses a data structure that is supplied from the QSYSINC library. When you use this data structure, you can prevent errors in data structure creation from happening. If the data structures change from release to release, updates to programs do not have to be done. The application program would have to be updated *only* if a new field was added to the data structure and you *wanted* to use the field. The copying of the QSYSINC data structure is shown at (2).

```
 *
 *
 ****************************************************************
 *
 *Program Name: PGM2
 *
 *Program Language:  RPG
 *
 *Description: This sample program illustrates the correct
 *             way of defining data structures.
 *
 *Header Files Included: QUSEC - Error Code Parameter
 *                       QUSGEN - User Space Format for Generic Header
 *                       QUSLJOB - List Job API
 *
 *APIs Used:  QUSCRTUS  - Create User Space
 *            QUSLJOB   - List Job
 *            QUSRTVUS  - Retrieve User Space
 *            QUSDLTUS  - Delete User Space
 *
 *
 * THIS PROGRAM WILL CREATE THE NECESSARY SPACE AND THEN CALL
 * THE QUSLJOB API TO GET A LIST OF ALL ACTIVE JOBS ON THE SYSTEM.
 * THE FIRST JOB NAME/USER WILL BE DISPLAYED TO THE USER.
 *
I/COPY QSYSINC/QRPGSRC,QUSGEN
I/COPY QSYSINC/QRPGSRC,QUSEC
I/COPY QSYSINC/QRPGSRC,QUSLJOB                              (2)
 ** JOB NAME STRUCTURE FOR CALLING QUSLJOB
IJOBNAM    DS
I I            '*ALL      '              1  10 JOB
I I            '*ALL      '             11  20 USER
I I            '*ALL'                   21  26 JOBNUM
 ** JOBL0100 FORMAT RETURNED FROM QUSLJOB API
 **
 **
ISPCNAM     DS
I I            'SPCNAME   '              1  10 SPC
I I            'QTEMP     '             11  20 LIB
 ** OTHER ASSORTED VARIABLES
I           DS
I I            2000                 B   1   40SIZ
I I                                 B   5   80START
I I                                 B   9  120LENDTA
I I            X'00'                   13  13 INTVAL
 *
 * SET UP TO ACCEPT EXCEPTIONS
C                     Z-ADD*ZEROS    QUSBNB
```

```
 *
 * CREATE THE SPACE TO HOLD THE DATA
C                   CALL 'QUSCRTUS'
C                   PARM           SPCNAM
C                   PARM 'EXT_ATTR'EXTATR 10
C                   PARM           SIZ
C                   PARM           INTVAL
C                   PARM '*ALL    'PUBAUT 10
C                   PARM 'TEXT DSC'TXTDSC 50
C                   PARM '*YES    'REPLAC 10
C                   PARM           QUSBN
 *
 * CALL THE API TO LIST THE ACTIVE JOBS
C                   CALL 'QUSLJOB'
C                   PARM           SPCNAM
C                   PARM 'JOBL0100'FORMAT  8
C                   PARM           JOBNAM
C                   PARM '*ACTIVE 'STAT   10
C                   PARM           QUSBN
 *
 * RETRIEVE THE OFFSET OF THE FIRST LIST ENTRY FROM THE SPACE
C                   Z-ADD1         START
C                   Z-ADD140       LENDTA
C                   CALL 'QUSRTVUS'
C                   PARM           SPCNAM
C                   PARM           START
C                   PARM           LENDTA
C                   PARM           QUSBP
C                   PARM           QUSBN
 *
 * RETRIEVE THE FIRST LIST ENTRY
C          QUSBPQ   ADD  1         START
C                   Z-ADD56        LENDTA
C                   CALL 'QUSRTVUS'
C                   PARM           SPCNAM
C                   PARM           START
C                   PARM           LENDTA
C                   PARM           QUSDD
C                   PARM           QUSBN
 *
 * DISPLAY THE JOB NAME
C                   DSPLY          QUSDDB
 *************************
 *  Correct job name     *
 *  will now show as     *
 *  'QCPF      '         *
 *************************
 * DELETE THE SPACE THAT HELD THE DATA
C                   CALL 'QUSDLTUS'
C                   PARM           SPCNAM
C                   PARM           QUSBN
 **
C                   SETON                     LR
```

Related concepts:

"Data types and APIs" on page 66
APIs support character data and binary data.

Defining receiver variables

When you define a receiver variable, the most common error is to create a receiver variable that is too small for the amount of data that it is to receive. Here are the program examples that show the incorrect and correct ways of defining receiver variables.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Incorrect program example: Defining receiver variables

The following example program might fail because the receiver variable has been defined as 50 bytes, as shown at (1), but 60 bytes are being requested to be passed back from the API, as shown at (2) in the incorrect program and at (3) in the correct program. The correct coding is shown at (4).

When this situation happens, other variables are overwritten with unintended data. This causes the other variables to be incorrect. For example, the first 10 characters of the QUSBN parameter can be written over with these extra characters. On the call to the next API, the error code parameter might appear to contain meaningless characters that can cause the next call to an API to fail.

```
 ****************************************************************
 *
 *Program Name: PGM1
 *
 *Program Language:  RPG
 *
 *Description: This sample program illustrates the incorrect
 *             way of defining receiver variables.
 *
 *Header Files Included: QUSEC   - Error Code Parameter
 *                       QUSLJOB - List Job API
 *                       QUSGEN  - User Space Format for Generic Header
 *
 *APIs Used:  QUSCRTUS - Create User Space
 *            QUSLJOB  - List Job
 *            QUSRTVUS - Retrieve User Space
 *            QUSDLTUS - Delete User Space
 ****************************************************************
 * THIS PROGRAM WILL CREATE THE NECESSARY SPACE AND THEN CALL
 * THE QUSLJOB API TO GET A LIST OF ALL ACTIVE JOBS ON THE SYSTEM.
 * BRING IN THE GENERIC USER SPACE HEADER FROM QSYSINC
I/COPY QSYSINC/QRPGSRC,QUSGEN
 *
 * BRING IN THE ERROR STRUCTURE FROM QSYSINC
I/COPY QSYSINC/QRPGSRC,QUSEC
 *
** JOBL0100 FORMAT RETURNED FROM QUSLJOB API
I/COPY QSYSINC/QRPGSRC,QUSLJOB
 *
 ** JOB NAME STRUCTURE FOR CALLING QUSLJOB
IJOBNAM     DS
I I           '*ALL      '              1  10 JOB
I I           '*ALL      '             11  20 USER
I I           '*ALL'                   21  26 JOBNUM
ISPCNAM     DS
I I           'SPCNAME   '              1  10 SPC
I I           'QTEMP     '             11  20 LIB
 ** OTHER ASSORTED VARIABLES
I           DS
I I           2000                 B   1   40SIZ
I                                  B   5   80START
I                                  B   9  120LENDTA
I I           X'00'                   13  13 INTVAL
 *
 * SET UP TO ACCEPT EXCEPTIONS
C                     Z-ADD*ZEROS    QUSBNB
 *
 * CREATE THE SPACE TO HOLD THE DATA
C                     CALL 'QUSCRTUS'
C                     PARM           SPCNAM
C                     PARM 'EXT_ATTR'EXTATR 10
C                     PARM           SIZ
C                     PARM           INTVAL
C                     PARM '*ALL    'PUBAUT 10
C                     PARM 'TEXT DSC'TXTDSC 50
```

```
C                   PARM '*YES    'REPLAC 10
C                   PARM           QUSBN
 *
 * CALL THE API TO LIST THE ACTIVE JOBS
C                   CALL 'QUSLJOB'
C                   PARM           SPCNAM
C                   PARM 'JOBL0100'FORMAT  8
C                   PARM           JOBNAM
C                   PARM '*ACTIVE 'STAT   10
C                   PARM           QUSBN
 *
 * RETRIEVE THE OFFSET OF THE FIRST LIST ENTRY FROM THE SPACE
C                   Z-ADD1         START
C                   Z-ADD140       LENDTA
C                   CALL 'QUSRTVUS'
C                   PARM           SPCNAM
C                   PARM           START
C                   PARM           LENDTA
C                   PARM           QUSBP
C                   PARM           QUSBN
 *
 * RETRIEVE THE LIST ENTRIES
C          QUSBPQ   ADD  1         START
 *
C                   Z-ADD60        LENDTA                (2)
 *
C                   Z-ADD1          X      90
C         X         DOWLEQUSBPS
C                   CALL 'QUSRTVUS'
C                   PARM           SPCNAM
C                   PARM           START
C                   PARM           LENDTA
C                   PARM           RECVR  50             (1)
C                   PARM           QUSBN
 *
C                   MOVEL  RECVR  QUSDD
 *
C                   ADD  QUSBPT    START
C                   ADD  1         X
C                   END
 * DELETE THE SPACE THAT HELD THE DATA
C                   CALL 'QUSDLTUS'
C                   PARM           SPCNAM
C                   PARM           QUSBN
 *
C                   SETON                     LR
```

Correct program example: Defining receiver variables

The following example program defines a larger receiver variable: 60 bytes. This is shown at position (4). This increase in the receiver variable allows up to 60 bytes of data to be received.

```
*****************************************************************
*
*Program Name: PGM2
*
*Program Language:  RPG
*
*Description: This sample program illustrates the correct
*             way of defining receiver variables.
*
*Header Files Included: QUSEC - Error Code Parameter
*                       QUSLJOB - List Job API
*                       QUSGEN  - User Space Format for Generic Header
*
*APIs Used:  QUSCRTUS - Create User Space
*            QUSLJOB  - List Job
```

```
 *            QUSRTVUS - Retrieve User Space
 *            QUSDLTUS - Delete User Space
 ****************************************************************
 *
 * BRING IN THE ERROR STRUCTURE FROM QSYSINC
I/COPY QSYSINC/QRPGSRC,QUSEC
 * BRING IN THE GENERIC USER SPACE HEADER FROM QSYSINC
I/COPY QSYSINC/QRPGSRC,QUSGEN
 *
 ** JOBL0100 FORMAT RETURNED FROM QUSLJOB API
I/COPY QSYSINC/QRPGSRC,QUSLJOB
 *
 ** JOB NAME STRUCTURE FOR CALLING QUSLJOB
IJOBNAM     DS
I I           '*ALL      '              1  10 JOB
I I           '*ALL      '             11  20 USER
I I           '*ALL'                   21  26 JOBNUM
ISPCNAM     DS
I I           'SPCNAME   '              1  10 SPC
I I           'QTEMP     '             11  20 LIB
 ** OTHER ASSORTED VARIABLES
I           DS
I I           2000                 B   1   40SIZ
I                                  B   5   80START
I                                  B   9  120LENDTA
I I           X'00'                   13  13 INTVAL
 *
 * SET UP TO ACCEPT EXCEPTIONS
C                     Z-ADD*ZEROS    QUSBNB
 *
 * CREATE THE SPACE TO HOLD THE DATA
C                     CALL 'QUSCRTUS'
C                     PARM           SPCNAM
C                     PARM 'EXT_ATTR'EXTATR 10
C                     PARM           SIZ
C                     PARM           INTVAL
C                     PARM '*ALL    'PUBAUT 10
C                     PARM 'TEXT DSC'TXTDSC 50
C                     PARM '*YES    'REPLAC 10
C                     PARM           QUSBN
 *
 * CALL THE API TO LIST THE ACTIVE JOBS
C                     CALL 'QUSLJOB'
C                     PARM           SPCNAM
C                     PARM 'JOBL0100'FORMAT  8
C                     PARM           JOBNAM
C                     PARM '*ACTIVE 'STAT   10
C                     PARM           QUSBN
 *
 * RETRIEVE THE OFFSET OF THE FIRST LIST ENTRY FROM THE SPACE
C                     Z-ADD1         START
C                     Z-ADD140       LENDTA
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM
C                     PARM           START
C                     PARM           LENDTA
C                     PARM           QUSBP
C                     PARM           QUSBN
 *
 * RETRIEVE LIST ENTRIES
C         QUSBPQ      ADD  1         START
 *
C                     Z-ADD60        LENDTA               (3)
*
C                     Z-ADD1          X      90
C         X           DOWLEQUSBPS
C                     CALL 'QUSRTVUS'
```

```
C                   PARM           SPCNAM
C                   PARM           START
C                   PARM           LENDTA
C                   PARM           RECVR  60             (4)
C                   PARM           QUSBN
 *
C                   MOVELRECVR     QUSDD
C                   ADD  QUSBPT    START
C                   ADD  1         X
C                   END
 * DELETE THE SPACE THAT HELD THE DATA
C                   CALL 'QUSDLTUS'
C                   PARM           SPCNAM
C                   PARM           QUSBN
 *
C                   SETON                     LR
 *
```

Defining list-entry format lengths

When you define the list-entry format length, the most common error is to statically encode the format length in your program. Here are the program examples that show the incorrect and correct ways of defining list-entry format lengths.

The program uses the format length to advance to the next list entry in the user space. The length of the format might change from release to release. Therefore, when the format length changes, your program could be susceptible to being pointed at an incorrect position in the user space and nonsensical data could be placed in the receiver variable.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Incorrect program example: Defining list entry format lengths

The program has the length of the list entry format hard coded. This is shown at (1). If your program runs on a Version 2 Release 2 system, that value would work. However, with Version 2 Release 3, the format size increased from 52 to 56 bytes. The correct coding is shown at (2).

```
 ****************************************************************
 *
 *Program Name: PGM1
 *
 *Program Language:  RPG
 *
 *Description: This sample program illustrates the incorrect
 *             way of using list entry length formats.
 *
 *Header Files Included: QUSEC - Error Code Parameter
 *                       QUSLJOB - List Job API
 *                       QUSGEN -  User Space Format for Generic Header
 *
 *APIs Used:  QUSCRTUS - Create User Space
 *            QUSLJOB  - List Job
 *            QUSRTVUS - Retrieve User Space
 *            QUSDLTUS - Delete User Space
 ****************************************************************
 *
 * THIS PROGRAM WILL CREATE THE NECESSARY SPACE AND THEN CALL
 * THE QUSLJOB API TO GET A LIST OF ALL ACTIVE JOBS ON THE SYSTEM.
 * THE FIRST JOB NAME/USER WILL BE DISPLAYED TO THE USER.
I/COPY QSYSINC/QRPGSRC,QUSGEN
I/COPY QSYSINC/QRPGSRC,QUSLJOB
 *
 * BRING IN THE ERROR STRUCTURE FROM QSYSINC
```

```
I/COPY QSYSINC/QRPGSRC,QUSEC
 *
 ** JOB NAME STRUCTURE FOR CALLING QUSLJOB
IJOBNAM     DS
I I           '*ALL      '              1  10 JOB
I I           '*ALL      '             11  20 USER
I I           '*ALL'                   21  26 JOBNUM
 * FORMAT JOBL0100 FOR QUSLJOB API
 *
 ** DATA STRUCTURE CONTAINING SPACE NAME/LIB
ISPCNAM     DS
I I           'SPCNAME   '              1  10 SPC
I I           'QTEMP     '             11  20 LIB
 ** OTHER ASSORTED VARIABLES
I           DS
I I           2000                 B   1   40SIZ
I                                  B   5   80START
I                                  B   9  120LENDTA
I I           X'00'                   13  13 INTVAL
 *
 * SET UP TO ACCEPT EXCEPTIONS
C                     Z-ADD*ZEROS    QUSBNB
 *
 * CREATE THE SPACE TO HOLD THE DATA
C                     CALL 'QUSCRTUS'
C                     PARM           SPCNAM
C                     PARM 'EXT_ATTR'EXTATR 10
C                     PARM           SIZ
C                     PARM           INTVAL
C                     PARM '*ALL    'PUBAUT 10
C                     PARM 'TEXT DSC'TXTDSC 50
C                     PARM '*YES    'REPLAC 10
C                     PARM           QUSBN
 *
 * CALL THE API TO LIST THE ACTIVE JOBS
C                     CALL 'QUSLJOB'
C                     PARM           SPCNAM
C                     PARM 'JOBL0100'FORMAT  8
C                     PARM           JOBNAM
C                     PARM '*ACTIVE 'STAT   10
C                     PARM           QUSBN
 *
 * RETRIEVE INFORMATION ABOUT THE USER SPACE AND ITS CONTENTS
C                     Z-ADD1         START
C                     Z-ADD140       LENDTA
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM
C                     PARM           START
C                     PARM           LENDTA
C                     PARM           QUSBP
C                     PARM           QUSBN
 *
 * RETRIEVE LIST ENTRIES
C         QUSBPQ      ADD  1         START
C                     Z-ADD52        LENDTA
C                     Z-ADD1         X               90
C           X         DOWLEQUSBPS
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM
C                     PARM           START
C                     PARM           LENDTA
C                     PARM           QUSDD
C                     PARM           QUSBN
 *
 * RETRIEVE THE NEXT LIST ENTRY (SPECIFYING LAST RELEASE'S
 * FORMAT LENGTH AS THE AMOUNT TO BUMP THE POINTER - THIS
 * WILL RESULT IN "GARBAGE" IN THE RECEIVER VARIABLE BECAUSE THE
```

```
 * FORMAT IS NOW 56 BYTES LONG)
 *
 * DISPLAY THE INFORMATION RETURNED
C                   MOVELQUSDD     RECVR  52
C                   DSPLY          RECVR
C                   ADD  52        START                    (1)
C                   ADD  1         X
C                   END
 *
 * DELETE THE SPACE THAT HELD THE DATA
C                   CALL 'QUSDLTUS'
C                   PARM           SPCNAM
C                   PARM           QUSBN
 *
C                   SETON                     LR
```

Correct program example: Defining list entry format lengths

The following program correctly uses the list entry length that is defined in the space header for the QUSRTVUS API to advance from one entry to the next. This is shown at (2). If you use this value in your program, you will always have the correct list entry length regardless of the version or release level of the API.

```
*****************************************************************
 *
 *Program Name: PGM2
 *
 *Program Language:  RPG
 *
 *Description: This sample program illustrates the correct
 *             way of using list entry length formats.
 *
 *Header Files Included: QUSEC - Error Code Parameter
 *                       QUSLJOB - List Job API
 *                       QUSGEN -  User Space Format for Generic Header
 *
 *APIs Used:  QUSCRTUS - Create User Space
 *            QUSLJOB  - List Job
 *            QUSRTVUS - Retrieve User Space
 *            QUSDLTUS - Delete User Space
 *****************************************************************
 *
 * THIS PROGRAM WILL CREATE THE NECESSARY SPACE AND THEN CALL
 * THE QUSLJOB API TO GET A LIST OF ALL ACTIVE JOBS ON THE SYSTEM.
 *
I/COPY QSYSINC/QRPGSRC,QUSGEN
I/COPY QSYSINC/QRPGSRC,QUSLJOB
I/COPY QSYSINC/QRPGSRC,QUSEC
 *
 ** JOB NAME STRUCTURE FOR CALLING QUSLJOB
IJOBNAM     DS
I I           '*ALL      '              1  10 JOB
I I           '*ALL      '             11  20 USER
I I           '*ALL'                   21  26 JOBNUM
 *
 ** DATA STRUCTURE TO HOLD SPACE NAME
ISPCNAM     DS
I I           'SPCNAME   '              1  10 SPC
I I           'QTEMP     '             11  20 LIB
 ** OTHER ASSORTED VARIABLES
I           DS
I I           2000                 B   1   40SIZ
I                                  B   5   80START
I                                  B   9  120LENDTA
I I           X'00'                   13  13 INTVAL
 *
```

```
 * SET UP TO ACCEPT EXCEPTIONS
C                   Z-ADD*ZEROS    QUSBNB
 *
 * CREATE THE SPACE TO HOLD THE DATA
C                   CALL 'QUSCRTUS'
C                   PARM           SPCNAM
C                   PARM 'EXT_ATTR'EXTATR 10
C                   PARM           SIZ
C                   PARM           INTVAL
C                   PARM '*ALL    'PUBAUT 10
C                   PARM 'TEXT DSC'TXTDSC 50
C                   PARM '*YES    'REPLAC 10
C                   PARM           QUSBN
 *
 * CALL THE API TO LIST THE ACTIVE JOBS
C                   CALL 'QUSLJOB'
C                   PARM           SPCNAM
C                   PARM 'JOBL0100'FORMAT  8
C                   PARM           JOBNAM
C                   PARM '*ACTIVE 'STAT   10
C                   PARM           QUSBN
 *
 * RETRIEVE INFORMATION ABOUT THE USER SPACE AND ITS CONTENTS
C                   Z-ADD1         START
C                   Z-ADD140       LENDTA
C                   CALL 'QUSRTVUS'
C                   PARM           SPCNAM
C                   PARM           START
C                   PARM           LENDTA
C                   PARM           QUSBP
C                   PARM           QUSBN
*
 * RETRIEVE THE FIRST LIST ENTRY BASED ON THE LIST ENTRY OFFSET
 * FOUND IN THE SPACE HEADER
C         QUSBPQ    ADD  1         START
C                   Z-ADD52        LENDTA
C                   Z-ADD1         X               90
C          X        DOWLEQUSBPS
C                   CALL 'QUSRTVUS'
C                   PARM           SPCNAM
C                   PARM           START
C                   PARM           LENDTA
C                   PARM           QUSDD
C                   PARM           QUSBN
 *
 * RETRIEVE THE NEXT LIST ENTRY (SPECIFYING LIST ENTRY LENGTH
 * RETRIEVED FROM THE SPACE HEADER)
C                   ADD  QUSBPT    START                      (2)
 *
 * DISPLAY THE INFORMATION RETURNED
C                   MOVELQUSDD     RECVR  52
C                   DSPLY          RECVR
C                   ADD  1         X
C                   END
 *
 * DELETE THE SPACE THAT HELD THE DATA
C                   CALL 'QUSDLTUS'
C                   PARM           SPCNAM
C                   PARM           QUSBN
 **
C                   SETON                     LR
```

Related reference:

"Example in OPM RPG: List APIs" on page 87
This OPM RPG program prints a report that shows all objects that adopt owner authority.

Using null pointers with program-based APIs

Many programmers, especially those with a C programming background, view ignored parameters and NULL parameters as being the same. This expectation can lead to unexpected results when program-based APIs are used. Here are the program examples that show the incorrect and correct ways of using null pointers with program-based APIs.

Note: Using NULL with ignored parameters is primarily a consideration with program-based APIs. APIs based on service programs allow you to pass NULL parameters to indicate omitted parameter values.

Even though the value assigned to a parameter is not used, the parameter itself must be addressable. When you use NULL for a parameter value, the system conceptually passes an address that can be equated with 0, where 0 indicates that the parameter cannot be addressed. This lack of addressability often results in a function check (MCH3601). Additionally, other error messages might also occur.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Incorrect program example: Using null pointers with program-based APIs

The following program has two parameter values coded as NULL. They are the ignored parameters of the member and record format used in the List Database Relations (QDBLDBR) API, which is shown at (1). The correct coding is shown at (2).

When the program is called, a machine function check of MCH3601 is reported because the address of the required parameters member and record format are specified as NULL.

```
/************************************************************/
/*                                                          */
/*Program Name: PGM1                                        */
/*                                                          */
/*Program Language:  ILE C                                  */
/*                                                          */
/*Description: This sample program illustrates the incorrect  */
/*             use of ignored and null parameters.          */
/*                                                          */
/*Header Files Included: <stdio.h>                    */
/*                       <qusec.h>                    */
/*                       <qusgen.h>                   */
/*                       <qdbldbr.h>                  */
/*                       <quscrtus.h>                 */
/*                       <qusptrus.h>                 */
/*                       <qliept.h>                   */
/*                                                          */
/*APIs Used:  QUSCRTUS - Create User Space                  */
/*            QDBLDBR  - List Database Relations            */
/*            QUSPTRUS - Retrieve Pointer to User Space     */
/************************************************************/
#include <stdio.h>
#include <qusec.h>
#include <qusgen.h>
#include <qdbldbr.h>
#include <quscrtus.h>
#include <qusptrus.h>
#include <qliept.h>
main()
 {

   /************************************************************/
   /* initialize program data elements                         */
   /************************************************************/
```

```
   char initial_value = 0x00;
   char text_description[50] =
         "test of QDBLDBR API                                ";
   char qualified_usrspc_name[20] = "GETLDBR   QTEMP     ";
   Qus_EC_t error_code;
   Qus_Generic_Header_0100_t *header_ptr;
   error_code.Bytes_Provided = 0;
   /*************************************************************/
   /* Create the user space to hold API results                 */
   /*************************************************************/

   QUSCRTUS(qualified_usrspc_name, "SPACE     ", 1,
            &initial_value, "*ALL      ", text_description,
            "*YES      ", &error_code, "*USER     ");
   /*************************************************************/
   /* Get list of file dependencies in current library          */
   /*                                                           */
   /* Note that in this API call NULL pointers are being        */
   /* used for the "ignored" parameters Member and              */
   /* Record_Format.  This convention is not valid as the       */
   /* parameters must address a valid storage address.          */
   /* The value                                                 */
   /* assigned to a storage location is not important, the      */
   /* passing of a valid storage location is.                   */
   /*                                                           */
   /* The next statement will cause a MCH3601                   */
   /*************************************************************/

   QDBLDBR(qualified_usrspc_name, "DBRL0100", "*ALL      *CURLIB   ",
           NULL, NULL, &error_code);                               (1)


   /*************************************************************/
   /* Get pointer to user space which contains dependencies     */
   /*************************************************************/

   QUSPTRUS(qualified_usrspc_name, &header_ptr, &error_code);

   /*************************************************************/
   /* and display number of entries generated                   */
   /*************************************************************/

   printf("The number of entries returned is %d\n",
           header_ptr->Number_List_Entries);
}
```

Correct program example: Using null pointers with program-based APIs

The following program specifies that blanks be used as the values for both the member and record format parameters. This coding is shown at (2) in the example program. By using blanks, the storage or address location of those parameters is identified and passed when needed.

```
/*************************************************************/
/*                                                           */
/*Program Name: PGM2                                         */
/*                                                           */
/*Program Language:  ILE C                                   */
/*                                                           */
/*Description: This sample program illustrates the correct   */
/*             use of ignored and null parameters.           */
/*                                                           */
/*Header Files Included: <stdio.h>                           */
/*                       <qusec.h>                           */
/*                       <qusgen.h>                          */
/*                       <qdbldbr.h>                         */
/*                       <quscrtus.h>                        */
/*                       <qusptrus.h>                        */
```

```
/*                      <qliept.h>                          */
/*                                                          */
/*APIs Used:  QUSCRTUS - Create User Space                  */
/*            QDBLDBR  - List Database Relations            */
/*            QUSPTRUS - Retrieve Pointer to User Space     */
/************************************************************/
#include <stdio.h>
#include <qusec.h>
#include <qusgen.h>
#include <qdbldbr.h>
#include <quscrtus.h>
#include <qusptrus.h>
#include <qliept.h>
main()
 {

   /************************************************************/
   /* initialize program data elements                         */
   /************************************************************/

   char initial_value = 0x00;
   char text_description[50] =
         "test of QDBLDBR API                                ";
   char qualified_usrspc_name[20] = "GETLDBR   QTEMP     ";
   Qus_EC_t error_code;
   Qus_Generic_Header_0100_t *header_ptr;
   error_code.Bytes_Provided = 0;

   /************************************************************/
   /* Create the user space to hold API results                */
   /************************************************************/

   QUSCRTUS(qualified_usrspc_name, "SPACE     ", 1,
            &initial_value, "*ALL      ", text_description,
            "*YES      ", &error_code, "*USER     ");

   /************************************************************/
   /* Get list of file dependencies in current library         */
   /*                                                          */
   /* Note that in this API call, blank characters are being   */
   /* used for the "ignored" parameters Member and             */
   /* Record_Format.  While the value is ignored, a valid      */
   /* parameter storage location must still be passed          */
   /************************************************************/

   QDBLDBR(qualified_usrspc_name, "DBRL0100", "*ALL      *CURLIB   ",
           " ", " ", &error_code);                                    (2)


   /************************************************************/
   /* Get pointer to user space which contains dependencies    */
   /************************************************************/

   QUSPTRUS(qualified_usrspc_name, &header_ptr, &error_code);

   /************************************************************/
   /* and display number of entries generated                  */
   /************************************************************/

   printf("The number of entries returned is %d\n",
           header_ptr->Number_List_Entries);
 }
```

Defining byte alignment

Correct byte alignment ensures that an API reads the data from the beginning of a record rather than at some other point. Here are the program examples that show the incorrect and correct ways of defining byte alignment.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Incorrect program example: Defining byte alignment

This program illustrates byte alignment while defining a structure. This is shown at (1). Four-byte alignment is required when using this program.

Variable-length records must begin on a 4-byte boundary. As shown at (1), the variable-length record CCSID_rec is not beginning on a 4-byte boundary. When the API accesses the CCSID_rec record, 4-byte alignment is forced by padding the first 3 bytes of the CCSID_rec between the replace field and the start of the CCSID_rec record. (2) shows that the variable-length record is not 4-byte aligned (the value is 13, which is not divisible by 4). The correct coding is shown at (3).

Note: Not all APIs require a 4-byte boundary. APIs based on service programs, such as the Add Exit Program (QusAddExitProgram) API, require a 4-byte boundary.

```
/*****************************************************************  */
/*                                                                  */
/*Program Name: PGM1                                                */
/*                                                                  */
/*Program Language:  ILE C                                          */
/*                                                                  */
/*Description:    This program illustrates improper byte            */
/*                alignment when using variable length              */
/*                records.                                          */
/*                                                                  */
/*                                                                  */
/*Header Files Included:  <stdio.h>                                 */
/*                        <signal.h>                                */
/*                        <string.h>                                */
/*                        <stdlib.h>                                */
/*                        <qusrgfa1.h>                              */
/*                        <qusec.h>                                 */
/*                        <qliept.h>                                */
/*                                                                  */
/*  APIs Used:     QusAddExitProgram    - Add an exit program       */
/*                                                                  */
/*******************************************************************/
/*******************************************************************/
/*                     Includes                                    */
/*******************************************************************/
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <qusrgfa1.h>
#include <qusec.h>
#include <qliept.h>
/*******************************************************************/
/*                     Structures                                  */
/*                                                                 */
/*******************************************************************/

typedef struct {                     /* Error code                 */
 Qus_EC_t ec_fields;
 char     exception_data[100];
} error_code_struct;
```

```
typedef struct {                     /* Exit program attribute keys*/
  int              num_rec;
  Qus_Vlen_Rec_4_t replace_rec;
  char             replace;
  Qus_Vlen_Rec_4_t CCSID_rec;                          (1)
  int              CCSID;
  Qus_Vlen_Rec_4_t desc_rec;
  char             desc[50];
} addep_attributes;
/*********************************************************************/
/*                                                                   */
/*                            main                                   */
/*                                                                   */
/*********************************************************************/
int main()
{
  error_code_struct error_code;
  addep_attributes attrib_keys;

  /*******************************************************************/
  /* Initialize the error code parameter.                            */
  /*******************************************************************/
  error_code.ec_fields.Bytes_Provided=sizeof(error_code_struct);

  /*******************************************************************/
  /* Set the total number of exit program attributes that we are     */
  /* specifying on the call.  We will let the API take the default   */
  /* for the attributes that we are not specifying.                  */
  /*******************************************************************/
  attrib_keys.num_rec=3;
  /*******************************************************************/
  /* Set the values for the three attributes that we will be         */
  /* specifying:                                                     */
  /*       Replace exit program    = 1     (CHAR(1) field)           */
  /*       Exit program data CCSID = 37    (BIN(4) field)            */
  /*       Exit program description='THIS IS A TEST EXIT PROGRAM'    */
  /*                                       (CHAR(50) field)          */
  /*                                                                 */
  /* The structure for the exit program attributes defined above is  */
  /* as follows:                                                     */
  /*                                                                 */
  /*      typedef struct {                                           */
  /*        int              num_rec;                                */
  /*        Qus_Vlen_Rec_4_t replace_rec;                            */
  /*        char             replace;                                */
  /*        Qus_Vlen_Rec_4_t CCSID_rec;                              */
  /*        int              CCSID;                                  */
  /*        Qus_Vlen_Rec_4_t desc_rec;                               */
  /*        char             desc[50];                               */
  /*      } addep_attributes;                                        */
  /*                                                                 */
  /* and the Qus_Vlen_Rec_4_t structure is defined in                 */
  /* qus.h (included by qusrgfa1) as:                                */
  /*                                                                 */
  /*      typedef _Packed struct Qus_Vlen_Rec_4 {                    */
  /*        int  Length_Vlen_Record;                                 */
  /*        int  Control_Key;                                        */
  /*        int  Length_Data;                                        */
  /*      **char Data[];-> this field is supplied by                 */
  /*        the user                                                 */
  /*      } Qus_Vlen_Rec_4_t;                                        */
  /*                                                                 */
  /* This structure is mapped in bytes as follows:                   */
  /*      {                                                          */
  /*       BIN(4)   - num_rec                                        */
  /*       BIN(4)   - length variable length record for replace key */
```

```
  /*       BIN(4)   - replace key                                    */
  /*       BIN(4)   - length replace data                            */
  /*       CHAR(1)  - replace data                                   */
  /*       BIN(4)   - length variable length record for CCSID key    */
  /*       BIN(4)   - CCSID key                                      */
  /*       BIN(4)   - length CCSID data                              */
  /*       BIN(4)   - CCSID data                                     */
  /*       BIN(4)   - length variable length record for description */
  /*                  key                                            */
  /*       BIN(4)   - description key                                */
  /*       BIN(4)   - length description key                         */
  /*       CHAR(50) - description data                               */
  /*      }                                                          */
  /*                                                                 */
  /*******************************************************************/
  attrib_keys.replace_rec.Length_Vlen_Record=13;          (2)
  attrib_keys.replace_rec.Control_Key=4;
  attrib_keys.replace_rec.Length_Data=1;
  attrib_keys.replace='1';

  attrib_keys.CCSID_rec.Length_Vlen_Record=16;
  attrib_keys.CCSID_rec.Control_Key=3;
  attrib_keys.CCSID_rec.Length_Data=4;
  attrib_keys.CCSID=37;

  attrib_keys.desc_rec.Length_Vlen_Record=39;
  attrib_keys.desc_rec.Control_Key=2;
  attrib_keys.desc_rec.Length_Data=27;
  memcpy(&attrib_keys.desc,
         "THIS IS A TEST EXIT PROGRAM",27);

  /*******************************************************************/
  /* Call the API to add the exit program.                           */
  /*******************************************************************/
  QusAddExitProgram("EXAMPLE_EXIT_POINT  ",
                    "EXMP0100",
                    1,
                    "EXAMPLEPGMEXAMPLELIB",
                    "EXAMPLE EXIT PROGRAM DATA",
                    25,
                    &attrib_keys,
                    &error_code);

  if (error_code.ec_fields.Bytes_Available != 0)
  {
     printf("ATTEMPT TO ADD AN EXIT PROGRAM FAILED WITH EXCEPTION:%.7s",
             error_code.ec_fields.Exception_Id);
     exit(1);
  }


} /* end program */
```

Correct program example: Defining byte alignment

The following example program shows a CHAR(3) bytes reserved field being added to the structure to maintain 4-byte alignment as shown at (4). This corresponds to (1) in the incorrect coding example. The 3 reserved bytes are included in the length of the replace variable-length record. (3) shows the variable-length record is now 4-byte aligned (record length of 16 is divisible by 4). This corresponds to (2) in the incorrect coding example.

```
/*******************************************************************/
/*                                                                 */
/*Program Name: PGM2                                               */
/*                                                                 */
/*Program Language:  ILE C                                         */
```

```
/*                                                                  */
/*Description:    This program illustrates proper byte              */
/*                alignment when using variable length              */
/*                records.                                          */
/*                                                                  */
/*                                                                  */
/*Header Files Included:  <stdio.h>                                 */
/*                        <signal.h>                                */
/*                        <string.h>                                */
/*                        <stdlib.h>                                */
/*                        <qusrgfa1.h>                              */
/*                        <qusec.h>                                 */
/*                        <qliept.h>                                */
/*                                                                  */
/*  APIs Used:     QusAddExitProgram    - Add an exit program       */
/*                                                                  */
/*                                                                  */
/********************************************************************/
/*                     Includes                                     */
/********************************************************************/
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <qusrgfa1.h>
#include <qusec.h>
#include <qliept.h>
/********************************************************************/
/*                     Structures                                   */
/********************************************************************/

typedef struct {                      /* Error code                 */
 Qus_EC_t ec_fields;
 char     exception_data[100];
} error_code_struct;

typedef struct {                      /* Exit program attribute keys*/
  int              num_rec;
  Qus_Vlen_Rec_4_t replace_rec;
  char             replace;
  char             Reserved[3];                        (4)
  Qus_Vlen_Rec_4_t CCSID_rec;
  int              CCSID;
  Qus_Vlen_Rec_4_t desc_rec;
  char             desc[100];
} addep_attributes;

/********************************************************************/
/*                                                                  */
/*                            main                                  */
/*                                                                  */
/********************************************************************/
int main()
{
  error_code_struct error_code;
  addep_attributes attrib_keys;

  /******************************************************************/
  /* Initialize the error code parameter.                           */
  /******************************************************************/
  error_code.ec_fields.Bytes_Provided=sizeof(error_code_struct);

  /******************************************************************/
  /* Set the total number of exit program attributes that we are    */
  /* specifying on the call.  We will let the API take the default  */
  /* for the attributes that we are not specifying.                 */
  /******************************************************************/
```

```
  attrib_keys.num_rec=3;

  /*****************************************************************/
  /* Set the values for the three attributes that we will be       */
  /* specifying:                                                   */
  /*       Replace exit program    = 1     (CHAR(1) field)         */
  /*       Exit program data CCSID = 37    (BIN(4) field)          */
  /*       Exit program description='THIS IS A TEST EXIT PROGRAM'  */
  /*                                       (CHAR(50) field)        */
  /*****************************************************************/
  attrib_keys.replace_rec.Length_Vlen_Record=16;            (3)
  attrib_keys.replace_rec.Control_Key=4;
  attrib_keys.replace_rec.Length_Data=1;
  attrib_keys.replace='1';

  attrib_keys.CCSID_rec.Length_Vlen_Record=16;
  attrib_keys.CCSID_rec.Control_Key=3;
  attrib_keys.CCSID_rec.Length_Data=4;
  attrib_keys.CCSID=37;

  attrib_keys.desc_rec.Length_Vlen_Record=39;
  attrib_keys.desc_rec.Control_Key=2;
  attrib_keys.desc_rec.Length_Data=27;
  memcpy(&attrib_keys.desc,"THIS IS A TEST EXIT PROGRAM",27);

  /*****************************************************************/
  /* Call the API to add the exit program.                         */
  /*****************************************************************/
  QusAddExitProgram("EXAMPLE_EXIT_POINT  ",
                    "EXMP0100",
                    1,
                    "EXAMPLEPGMEXAMPLELIB",
                    "EXAMPLE EXIT PROGRAM DATA",
                    25,
                    &attrib_keys,
                    &error_code);

  if (error_code.ec_fields.Bytes_Available != 0)
  {
     printf("ATTEMPT TO ADD AN EXIT PROGRAM FAILED WITH EXCEPTION: %.7s",
             error_code.ec_fields.Exception_Id);
     exit(1);
  }


} /* end program */
```

Related concepts:

"Receiver variables" on page 73
A *receiver variable* is a program variable that is used as an output field to contain information that is returned from a retrieve API.

Using offsets in a user space

An offset indicates the point in a data structure where specific data should start. If you use offsets correctly, your program can extract specific pieces of data from the structure. Here are the program examples that show the incorrect and correct ways of using offsets.

Using offsets incorrectly can produce errors when coding in a base 1 language such as RPG and COBOL. One way to determine the base of a language is to determine how the first element of an array is specified. In a base 0 language, the first element is number 0. In base 1 languages, the first element is number 1.

The example programs in the following topics are coded using RPG. RPG is a base 1 language. However, APIs produce information using a base of 0. To compensate, the API user must add 1 to all decimal and hexadecimal offsets to the formats.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Incorrect program example: Using offsets in a user space

The beginning point for reading a user space is shown at (1). The data is read and placed into a user space. However, the data in the user space is incorrect because the position to start was off by 1. This program started to retrieve the data one character (or position) too soon. The correct coding is shown at (2).

```
I*****************************************************************
I*****************************************************************
I*
I*Program Name: APIUG1
I*
I*Programming Language:  RPG
I*
I*Description: This sample program illustrates the incorrect
I*             way of using the offset in a user space.
I*
I*Header Files Included: QUSGEN - Generic Header of a User Space
I*                       QUSEC - Error Code Parameter
I*                               (Copied into Program)
I*                       QUSLOBJ - List Objects API
I*
I*APIs Used:  QUSCRTUS - Create User Space
I*            QUSLOBJ  - List Objects
I*            QUSRTVUS - Retrieve User Space
I*            QUSDLTUS - Delete User Space
I*****************************************************************
I*****************************************************************
I*
I* Generic Header of a User Space Include
I*
I/COPY QSYSINC/QRPGSRC,QUSGEN
I*
I* Error Code Parameter Include for the APIs
I*
I* The following QUSEC include is copied into this program
I* so that the variable length field can be defined as a
I* fixed length.
I*
I*** START HEADER FILE SPECIFICATIONS ****************************
I*
I*Header File Name: H/QUSEC
I*
I*Descriptive Name: Error Code Parameter.
I*
I*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
I*All rights reserved.
I*US Government Users Restricted Rights -
I*Use, duplication or disclosure restricted
I*by GSA ADP Schedule Contract with IBM Corp.
I*
I*Licensed Materials-Property of IBM
I*
I*
I*Description: Include header file for the error code parameter.
I*
I*Header Files Included: None.
I*
```

```
I*Macros List: None.
I*
I*Structure List: Qus_EC_t
I*
I*Function Prototype List: None.
I*
I*Change Activity:
I*
I*CFD List:
I*
I*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
I*---- ------------ ----- ------ --------- ----------------------
I*$A0= D2862000     3D10  931201 DPOHLSON: New Include
I*
I*End CFD List.
I*
I*Additional notes about the Change Activity
I*End Change Activity.
I*** END HEADER FILE SPECIFICATIONS *****************************
I*****************************************************************
I*Record structure for Error Code Parameter
I****                                                          ***
I*NOTE: The following type definition only defines the corrected
I*   portion of the format.  Varying length field Exception
I*   Data will not be defined here.
I*****************************************************************
IQUSBN       DS
I*                                            Qus EC
I                                     B   1   40QUSBNB
I*                                            Bytes Provided
I                                     B   5   80QUSBNC
I*                                            Bytes Available
I                                         9  15 QUSBND
I*                                            Exception Id
I                                        16  16 QUSBNF
I*                                            Reserved
I*                                       17  17 QUSBNG
I*
I*                                       Varying length
I                                        17 100 QUSBNG
I*
I* List Objects API Include
I*
I/COPY QSYSINC/QRPGSRC,QUSLOBJ
I*
I* Qualified User Space Data Structure
I*
IUSERSP      DS
I I            'APIUG1    '               1  10 USRSPC
I I            'QGPL      '              11  20 SPCLIB
I* Qualified Object Name Data Structure
IOBJECT      DS
I I            '*ALL      '               1  10 OBJNAM
I I            'QGPL      '              11  20 OBJLIB
I*
I* Miscellaneous Data Structure
I*
I            DS
I* Set up parameters for the Create User Space API
I I            'TESTUSRSPC'                1  10 EXTATR
I I            X'00'                      11  11 INTVAL
I                                         12  12 RSVD1
I I            256                     B  13  160INTSIZ
I I            '*USE      '               17  26 PUBAUT
I I            'TEXT DESCRIPTION -        27  76 TEXT
I              'FOR USER SPACE -
I              'CALLED APIUG1  '
```

```
I I            '*YES      '             77  87 REPLAC
I* Set up parameters for the List Objects API
I I            'OBJL0100'               88  95 FORMAT
I I            '*ALL      '             96 105 OBJTYP
I                                      106 108 RSVD2
I* Set up parameters for the Retrieve User Space API
I I            1                     B 109 1120STRPOS
I I            192                   B 113 1160LENDTA
I                                    B 117 1200COUNT
C*
C* Create a user space called APIUG1 in library QGPL.
C*
C                     Z-ADD100       QUSBNB
C                     CALL 'QUSCRTUS'
C                     PARM           USERSP
C                     PARM           EXTATR
C                     PARM           INTSIZ
C                     PARM           INTVAL
C                     PARM           PUBAUT
C                     PARM           TEXT
C                     PARM           REPLAC
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C* Get a list of all objects in the QGPL library.
C*
C                     CALL 'QUSLOBJ'
C                     PARM           USERSP
C                     PARM           FORMAT
C                     PARM           OBJECT
C                     PARM           OBJTYP
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C* Look at the generic header.
C* The generic header contains information
C* about the list data section that is needed when processing
C* the entries.
C*
C                     CALL 'QUSRTVUS'
C                     PARM           USERSP
C                     PARM           STRPOS
C                     PARM           LENDTA
C                     PARM           QUSBP
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C* Check the information status field, QUSBPJ, to see if
C* the API was able to return all the information.
C* Possible values are:
C*   C -- Complete and accurate
C*   P -- Partial but accurate
C*   I -- Incomplete
C*
C           QUSBPJ    IFEQ 'C'
C           QUSBPJ    OREQ 'P'
C*
C* Check to see if any entries were put into the user space.
C*
C           QUSBPS    IFGT 0
C                     Z-ADD1         COUNT
C                     Z-ADDQUSBPQ    STRPOS                    (1)
C                     Z-ADD30        LENDTA
C* Walk through all the entries in the user space.
```

```
C           COUNT     DOWLEQUSBPS
C                     CALL 'QUSRTVUS'
C                     PARM           USERSP
C                     PARM           STRPOS
C                     PARM           LENDTA
C                     PARM           QUSDM
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C*
C* Process the objects.
C*
C                     ADD  1         COUNT
C                     ADD  QUSBPT    STRPOS
C                     ENDDO
C                     ENDIF
C*
C* Information in the user space is not accurate
C*
C                     ENDIF
C*
C* Delete the user space called APIUG1 in library QGPL.
C*
C                     CALL 'QUSDLTUS'
C                     PARM           USERSP
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C                     SETON                     LR
C                     RETRN
C*
C* End of MAINLINE
C*
C* Subroutine to handle errors returned in the error code
C* parameter.
C*
C           ERRCOD    BEGSR
C           QUSBNC    IFGT 0
C*
C* Process errors returned from the API.
C*
C                     END
C                     ENDSR
```

Correct program example: Using offsets in a user space

The following example program has code in it that compensates for the API offset convention of that starts at 0. The code adds 1 to the starting position (STRPOS) offset. This is shown at (2).

```
I*
I*Program Name: APIUG2
I*
I*Programming Language:  RPG
I*
I*Description: This sample program illustrates the correct
I*             way of using offsets in user space.
I*
I*Header Files Included: QUSGEN - Generic Header of a User Space
I*                       QUSEC - Error Code Parameter
I*                               (Copied into Program)
I*                       QUSLOBJ - List Objects API
I*
I*APIs Used:  QUSCRTUS - Create User Space
I*            QUSLOBJ  - List Objects
I*            QUSRTVUS - Retrieve User Space
```

```
I*            QUSDLTUS - Delete User Space
I*****************************************************************
I*****************************************************************
I*
I* Generic Header of a User Space Include
I*
I/COPY QSYSINC/QRPGSRC,QUSGEN
I*
I* Error Code Parameter Include for the APIs
I*
I* The following QUSEC include is copied into this program
I* so that the variable length field can be defined as a
I* fixed length.
I*
I*** START HEADER FILE SPECIFICATIONS ****************************
I*
I*Header File Name: H/QUSEC
I*
I*Descriptive Name: Error Code Parameter.
I*
I*5763-SS1, 5722-SS1  (C) Copyright IBM Corp. 1994, 2001
I*All rights reserved.
I*US Government Users Restricted Rights -
I*Use, duplication or disclosure restricted
I*by GSA ADP Schedule Contract with IBM Corp.
I*
I*Licensed Materials-Property of IBM
I*
I*
I*Description: Include header file for the error code parameter.
I*
I*Header Files Included: None.
I*
I*Macros List: None.
I*
I*Structure List: Qus_EC_t
I*
I*Function Prototype List: None.
I*
I*Change Activity:
I*
I*CFD List:
I*
I*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
I*---- ------------ ----- ------ --------- ----------------------
I*$A0= D2862000     3D10  931201 DPOHLSON: New Include
I*
I*End CFD List.
I*
I*Additional notes about the Change Activity
I*End Change Activity.
I*** END HEADER FILE SPECIFICATIONS ******************************
I*****************************************************************
I*Record structure for Error Code Parameter
I****                                                           ***
I*NOTE: The following type definition only defines the corrected
I*   portion of the format.  Varying length field Exception
I*   Data will not be defined here.
I*****************************************************************
IQUSBN       DS
I*                                          Qus EC
I                                    B   1   40QUSBNB
I*                                          Bytes Provided
I                                    B   5   80QUSBNC
I*                                          Bytes Available
I                                        9  15 QUSBND
I*                                          Exception Id
```

```
I                                       16  16 QUSBNF
I*                                         Reserved
I*                                      17  17 QUSBNG
I*
I*                                      Varying length
I                                       17 100 QUSBNG
I*
I* List Objects API Include
I*
I/COPY QSYSINC/QRPGSRC,QUSLOBJ
I*
I* Qualified User Space Data Structure
I*
IUSERSP      DS
I I             'APIUG1    '             1  10 USRSPC
I I             'QGPL      '            11  20 SPCLIB
I* Qualified Object Name Data Structure
IOBJECT      DS
I I             '*ALL      '             1  10 OBJNAM
I I             'QGPL      '            11  20 OBJLIB
I*
I* Miscellaneous Data Structure
I*
I            DS
I* Set up parameters for the Create User Space API
I I             'TESTUSRSPC'              1  10 EXTATR
I I             X'00'                    11  11 INTVAL
I                                        12  12 RSVD1
I I             256                   B  13  160INTSIZ
I I             '*USE      '             17  26 PUBAUT
I I             'TEXT DESCRIPTION -      27  76 TEXT
I               'FOR USER SPACE -
I               'CALLED APIUG2  '
I I             '*YES      '             77  87 REPLAC
I* Set up parameters for the List Objects API
I I             'OBJL0100'               88  95 FORMAT
I I             '*ALL      '             96 105 OBJTYP
I                                       106 108 RSVD2
I* Set up parameters for the Retrieve User Space API
I I             1                     B 109 1120STRPOS
I I             192                   B 113 1160LENDTA
I                                     B 117 1200COUNT
C*
C* Create a user space called APIUG1 in library QGPL.
C*
C                     Z-ADD100       QUSBNB
C                     CALL 'QUSCRTUS'
C                     PARM           USERSP
C                     PARM           EXTATR
C                     PARM           INTSIZ
C                     PARM           INTVAL
C                     PARM           PUBAUT
C                     PARM           TEXT
C                     PARM           REPLAC
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C* Get a list of all objects in the QGPL library.
C*
C                     CALL 'QUSLOBJ'
C                     PARM           USERSP
C                     PARM           FORMAT
C                     PARM           OBJECT
C                     PARM           OBJTYP
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
```

```
C                     EXSR ERRCOD
C*
C* Look at the generic header. This contains information
C* about the list data section that is needed when processing
C* the entries.
C*
C                     CALL 'QUSRTVUS'
C                     PARM           USERSP
C                     PARM           STRPOS
C                     PARM           LENDTA
C                     PARM           QUSBP
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C*
C* Check the information status field, QUSBPJ, to see if the
C* API was able to return all the information. Possible values
C* are:  C -- Complete and accurate
C*       P -- Partial but accurate
C*       I -- Incomplete.
C*
C           QUSBPJ    IFEQ 'C'
C           QUSBPJ    OREQ 'P'
C*
C* Check to see if any entries were put into the user space.
C*
C           QUSBPS    IFGT 0
C                     Z-ADD1         COUNT
C* Because RPG is Base 1, the offset must be increased by one.
C*
C           QUSBPQ    ADD  1         STRPOS                      (2)
C                     Z-ADD30        LENDTA
C* Walk through all the entries in the user space.
C           COUNT     DOWLEQUSBPS
C                     CALL 'QUSRTVUS'
C                     PARM           USERSP
C                     PARM           STRPOS
C                     PARM           LENDTA
C                     PARM           QUSDM
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C*
C* Process the objects.
C*
C                     ADD  1         COUNT
C                     ADD  QUSBPT    STRPOS
C                     ENDDO
C                     ENDIF
C*
C* Information in the user space is not accurate.
C*
C                     ENDIF
C*
C*
C* Delete the user space called APIUG1 in library QGPL.
C*
C                     CALL 'QUSDLTUS'
C                     PARM           USERSP
C                     PARM           QUSBN
C* See if any errors were returned in the error code parameter.
C                     EXSR ERRCOD
C*
C                     SETON                     LR
C                     RETRN
```

```
C*
C* End of MAINLINE
C*
C* Subroutine to handle errors returned in the error code
C* parameter.
C*
C           ERRCOD    BEGSR
C           QUSBNC    IFGT 0
C*
C* Process errors returned from the API.
C*
C                     END
C                     ENDSR
```

Coding for new functions

A new function from IBM can cause programs to fail if the programs do not allow for the handling of this function. Here are the program examples that show the incorrect and correct ways of using a new function.

Suppose that a new object type *SRVPGM is introduced, which can adopt owner authority.

A general theme of this example is never to assume that the values returned by an API are static. The IBM i operating system is continually evolving. While the example is based on the addition of a new object type, this philosophy should be applied to any output of an API. For example, if an API today can return *YES or *NO, you need to discretely check for these values because *MAYBE might be valid in the future. Similarly, if your application assumes that a particular integer output has a positive nonzero value (an offset for instance), you need to check for a positive nonzero value because future releases might return a negative value to indicate the new function.

Note: By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 571.

Incorrect program example: Coding for new functions

In this example program, a check is made to determine the object type. This is shown at (1). The example program considers only object types of *SQLPKG or *PGMs. This is because they are the only object types that could adopt owner authority before Version 2 Release 3. Since that time, a new object type of *SRVPGM has been introduced. *SRVPGM can adopt owner authority. Hence, this example program processes *SRVPGM objects as if they were *PGM objects. The correct coding is shown at (2).

```
D*****************************************************************
D*
D*Program Name: PGM1
D*
D*Program Language:  ILE RPG
D*
D*Description: This example program demonstrates how a program can
D*             be "broken" by new functions introduced on the system.
D*
D*
D*
D*Header Files Included: QUSGEN -   Generic Header of a User Space
D*                                   (Copied Into Program)
D*                       QUSEC -    Error Code Parameter
D*                                   (Copied Into Program)
D*                       QSYLOBJP - List Objects API
D*                                   (Copied Into Program)
D*
D*APIs Used:  QUSCRTUS - Create User Space
D*            QSYLOBJP - List Objects That Adopt Owner Authority
D*            QUSROBJD - Retrieve Object Description
D*            QUSPTRUS - Retrieve Pointer to User Space
```

```
D*****************************************************************
D*****************************************************************
D*
D* This program demonstrates how a program can be "broken" by
C* new functions introduced on the system.
D*
D*****************************************************************
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
DSPC_NAME         S             20    INZ('ADOPTS    QTEMP     ')
DSPC_SIZE         S              9B 0 INZ(1)
DSPC_INIT         S              1    INZ(X'00')
DLSTPTR           S               *
DSPCPTR           S               *
DARR              S              1    BASED(LSTPTR) DIM(32767)
DRCVVAR           S              8
DRCVVARSIZ        S              9B 0 INZ(%SIZE(RCVVAR))
D*****************************************************************
D*
D* The following QUSGEN include is copied into this program so
D* that it can be declared as BASED on SPCPTR, as shown at (3)
D* in the incorrect programs and at (4) in the correct program.
D*
D*****************************************************************
D*
D*Header File Name:  H/QUSGEN
D*
D*Descriptive Name:  Format structures for User Space for ILE/C
D*
D*5763-SS1, 5722-SS1  (C) Copyright IBM Corp.  1994, 2001
D*All rights reserved.
D*US Government Users Restricted Rights -
D*Use, duplication or disclosure restricted
D*by GSA ADP Schedule Contract with IBM Corp.
D*
D*Description: Contains the Generic Record format headers
D*          for the user space.
D*
D*Header Files Included: none.
D*
D*Macros List: none.
D*
D*Structure List: Qus_Generic_Header_0100
D*             Qus_Generic_Header_0300
D*
D*Function Prototype List: none.
D*
D*Change Activity:
D*
D*CFD List:
D*
D*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
D*---- ------------ ----- ------ --------- ----------------------
D*$A0= D2862000     3D10  940213 LUPA:     New Include
D*End CFD List.
D*
D*Additional notes about the Change Activity
D*End Change Activity.
D*** END HEADER FILE SPECIFICATIONS *****************************
D*****************************************************************
D*Type Definition for the User Space Generic Header.
D*****************************************************************
DQUSH0100         DS                  BASED(SPCPTR)                   (3)
D*                                              Qus Generic Header 0100
D QUSUA                   1     64
D*                                              User Area
D QUSSGH                 65     68B 0
```

```
D*                                              Size Generic Header
D QUSSRL                  69     72
D*                                              Structure Release Level
D QUSFN                   73     80
D*                                              Format Name
D QUSAU                   81     90
D*                                              API Used
D QUSDTC                  91    103
D*                                              Date Time Created
D QUSIS                  104    104
D*                                              Information Status
D QUSSUS                 105    108B 0
D*                                              Size User Space
D QUSOIP                 109    112B 0
D*                                              Offset Input Parameter
D QUSSIP                 113    116B 0
D*                                              Size Input Parameter
D QUSOHS                 117    120B 0
D*                                              Offset Header Section
D QUSSHS                 121    124B 0
D*                                              Size Header Section
D QUSOLD                 125    128B 0
D*                                              Offset List Data
D QUSSLD                 129    132B 0
D*                                              Size List Data
D QUSNBRLE               133    136B 0
D*                                              Number List Entries
D QUSSEE                 137    140B 0
D*                                              Size Each Entry
D QUSSIDLE               141    144B 0
D*                                              CCSID List Ent
D QUSCID                 145    146
D*                                              Country ID
D QUSLID                 147    149
D*                                              Language ID
D QUSSLI                 150    150
D*                                              Partial List Indicator
D QUSERVED00             151    192
D*                                              Reserved
D*****************************************************************
D*
D* The following QSYLOBJP include is copied into this program so
D* that it can be declared as BASED on LSTPTR, as shown at (5)
D* in the incorrect coding and (6) in the correct coding.
D*
D*****************************************************************
D*** START HEADER FILE SPECIFICATIONS ****************************
D*
D*Header File Name: H/QSYLOBJP
D*
D*Descriptive Name: List Objects That Adopt Owner Authority.
D*
D*
D*Description: Include header file for the QSYLOBJP API.
D*
D*Header Files Included: H/QSYLOBJP
D*                       H/QSY
D*
D*Macros List: None.
D*
D*Structure List: OBJP0100
D*              OBJP0200
D*              Qsy_OBJP_Header
D*
D*Function Prototype List: QSYLOBJP
D*
D*Change Activity:
```

```
D*
D*CFD List:
D*
D*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
D*---- ------------ ----- ------ --------- ----------------------
D*$A0= D2862000     3D10  931222 XZY0432:  New Include
D*
D*End CFD List.
D*
D*Additional notes about the Change Activity
D*End Change Activity.
D*** END HEADER FILE SPECIFICATIONS *****************************
D*****************************************************************
D*Prototype for calling Security API QSYLOBJP
D*****************************************************************
D QSYLOBJP        C                   'QSYLOBJP'
D*****************************************************************
D*Header structure for QSYLOBJP
D*****************************************************************
DQSYOBJPH         DS                  BASED(LSTPTR)                (5)
D*                                            Qsy OBJP Header
D QSYUN00                 1     10
D*                                  User name
D QSYCV00                11     30
D*                                  Continuation Value
D*****************************************************************
D*Record structure for OBJP0100 format
D*****************************************************************
DQSY0100L02       DS                  BASED(LSTPTR)                (5)
D*                                            Qsy OBJP0100 List
D  QSYNAME05              1     10
D  QSYBRARY05            11     20
D*                                  Qualified object name
D QSYOBJT12              21     30
D*                                  Object type
D QSYOBJIU               31     31
D*                                  Object in use
C*
C* Start of mainline
C*
C                   EXSR      INIT
C                   EXSR      PROCES
C                   EXSR      DONE
C*
C* Start of subroutines
C*
C*****************************************************************
C     PROCES        BEGSR
C*
C* This subroutine processes each entry returned by QSYLOBJP
C*
C*
C* Do until the list is complete
C*
C                   MOVE      QUSIS         LST_STATUS        1
C     LST_STATUS    DOUEQ     'C'
C*
C* If valid information was returned
C*
C     QUSIS         IFEQ      'C'
C     QUSIS         OREQ      'P'
C*
C* and list entries were found
C*
C     QUSNBRLE      IFGT      0
C*
C* set LSTPTR to the first byte of the user space
```

```
C*
C                   EVAL      LSTPTR = SPCPTR
C*
C* increment LSTPTR to the first list entry
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSOLD + 1))
C*
C* and process all of the entries
C*
C                   DO        QUSNBRLE
C     QSYOBJT12     IFEQ      '*SQLPKG'
C*
C* Process *SQLPKG type
C*
C                   ELSE                                              (1)
C*                                                                    |
C* This 'ELSE' logic is the potential bug in this program.  In        |
C* releases prior to V2R3 only *SQLPKGs and *PGMs could adopt         |
C* owner authority, and this program is assuming that if the         |
C* object type is not *SQLPKG then it must be a *PGM.  In V2R3       |
C* a new type of object (the *SRVPGM) was introduced.  As this       |
C* program is written, all *SRVPGMs that adopt the owner profile     |
C* will be processed as if they were *PGMs -- this erroneous         |
C* processing could definitely cause problems.                       |
C*                                                                    |
C     QSYNAME05     DSPLY                                           |
C                   END                                             V
C*
C* after each entry, increment LSTPTR to the next entry
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSSEE + 1))
C                   END
C                   END
C*
C* When all entries in this user space have been processed, check
C* if more entries exist than can fit in one user space
C*
C     QUSIS         IFEQ      'P'
C*
C* by resetting LSTPTR to the start of the user space
C*
C                   EVAL      LSTPTR = SPCPTR
C*
C* and then incrementing LSTPTR to the input parameter header
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSOIP + 1))
C*
C* If the continuation handle in the input parameter header is
C* blank, then set the list status to Complete
C*
C     QSYCV00       IFEQ      *BLANKS
C                   MOVE      'C'           LST_STATUS
C                   ELSE
C*
C* Else, call QSYLOBJP reusing the User Space to get more
C* List entries
C*
C                   MOVE      QSYCV00       CONTIN_HDL
C                   EXSR      GETLST
C                   MOVE      QUSIS         LST_STATUS
C                   END
C                   END
C                   ELSE
C*
C* And if an unexpected status, log an error (not shown) and exit
C*
C                   EXSR      DONE
```

```
C                   END
C                   END
C                   ENDSR
C*****************************************************************
C     GETLST        BEGSR
C*
C* Call QSYLOBJP to generate a list
C* The continuation handle is set by the caller of this subroutine.
C*
C                   CALL      QSYLOBJP
C                   PARM                    SPC_NAME
C                   PARM      'OBJP0100'    MBR_LIST          8
C                   PARM      '*CURRENT'    USR_PRF          10
C                   PARM      '*ALL'        OBJ_TYPE         10
C                   PARM                    CONTIN_HDL       20
C                   PARM                    QUSEC
C*
C* Check for errors on QSYLOBJP
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QSYLOBJP'    APINAM           10
C                   EXSR      APIERR
C                   END
C                   ENDSR
C*****************************************************************
C     INIT          BEGSR
C*
C* One-time initialization code for this program
C*
C* Set error code structure to not use exceptions
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Check to see if the user space was previously created in
C* QTEMP.  If it was, simply reuse it.
C*
C                   CALL      'QUSROBJD'
C                   PARM                    RCVVAR
C                   PARM                    RCVVARSIZ
C                   PARM      'OBJD0100'    ROBJD_FMT         8
C                   PARM                    SPC_NAME
C                   PARM      '*USRSPC'     OBJ_TYPE         10
C                   PARM                    QUSEC
C*
C* Check for errors on QUSROBJD
C*
C     QUSBAVL       IFGT      0
C*
C* If CPF9801, then user space was not found
C*
C     QUSEI         IFEQ      'CPF9801'
C*
C* So create a user space for the list generated by QSYLOBJP
C*
C                   CALL      'QUSCRTUS'
C                   PARM                    SPC_NAME
C                   PARM      'QSYLOBJP  '  EXT_ATTR         10
C                   PARM                    SPC_SIZE
C                   PARM                    SPC_INIT
C                   PARM      '*ALL'        SPC_AUT          10
C                   PARM      *BLANKS       SPC_TEXT         50
C                   PARM      '*YES'        SPC_REPLAC       10
C                   PARM                    QUSEC
C                   PARM      '*USER'       SPC_DOMAIN       10
C*
C* Check for errors on QUSCRTUS
C*
```

```
C     QUSBAVL       IFGT      0
C                   MOVEL     'QUSCRTUS'    APINAM          10
C                   EXSR      APIERR
C                   END
C*
C* Else, an error occurred accessing the user space
C*
C                   ELSE
C                   MOVEL     'QUSROBJD'    APINAM          10
C                   EXSR      APIERR
C                   END
C                   END
C*
C* Set QSYLOBJP (using GETLST) to start a new list
C*
C                   MOVE      *BLANKS       CONTIN_HDL
C                   EXSR      GETLST
C*
C* Get a resolved pointer to the user space for performance
C*
C                   CALL      'QUSPTRUS'
C                   PARM                    SPC_NAME
C                   PARM                    SPCPTR
C                   PARM                    QUSEC
C*
C* Check for errors on QUSPTRUS
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QUSPTRUS'    APINAM          10
C                   EXSR      APIERR
C                   END
C                   ENDSR
C*****************************************************************
C     APIERR        BEGSR
C*
C* Log any error encountered, and exit the program
C*
C     APINAM        DSPLY                   QUSEI
C                   EXSR      DONE
C                   ENDSR
C*****************************************************************
C     DONE          BEGSR
C*
C* Exit the program
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C                   ENDSR
```

Correct program example: Coding for new functions

In the following example program, code has been written that checks for object types *SRVPGM, *PGM, and *SQLPKG. If an object type is encountered that is unknown (it does not match *SRVPGM, *PGM, or *SQLPKG), an error is logged and an exit from the program takes place.

The coding to handle the integration of new function (in this case the new object type that can adopt owner authority) is shown at (2).

```
C*****************************************************************
C*
C*Program Name: PGM2
C*
C*Program Language:  ILE RPG
C*
C*Description: This example program demonstrates how a program can
C*             be coded to accept new functions introduced on the system.
```

```
C*
C*
C*
C*Header Files Included: QUSGEN -   Generic Header of a User Space
D*                                   (Copied Into Program)
C*                       QUSEC -    Error Code Parameter
D*                                   (Copied Into Program)
C*                       QSYLOBJP - List Objects API
D*                                   (Copied Into Program)
C*
C*APIs Used:  QUSCRTUS - Create User Space
C*            QSYLOBJP - List Objects That Adopt Owner Authority
C*            QUSROBJD - Retrieve Object Description
C*            QUSPTRUS - Retrieve Pointer to User Space
C*****************************************************************
H
C*****************************************************************
C*
D/COPY QSYSINC/QRPGLESRC,QUSEC
D*
DSPC_NAME         S             20    INZ('ADOPTS    QTEMP     ')
DSPC_SIZE         S              9B 0 INZ(1)
DSPC_INIT         S              1    INZ(X'00')
DLSTPTR           S               *
DSPCPTR           S               *
DARR              S              1    BASED(LSTPTR) DIM(32767)
DRCVVAR           S              8
DRCVVARSIZ        S              9B 0 INZ(%SIZE(RCVVAR))
D*****************************************************************
D*
D* The following QUSGEN include is copied into this program so
D* that it can be declared as BASED on SPCPTR, as shown at (3)
D* in the incorrect program and at (4) in the correct program.
D*
D*****************************************************************
D*
D*** START HEADER FILE SPECIFICATIONS ****************************
D*
D*Header File Name: H/QUSGEN
D*
D*Descriptive Name: Format structures for User Space for ILE/C
D*
D*
D*5763-SS1, 5722-SS1  (C) Copyright IBM Corp.  1994, 2001
D*All rights reserved.
D*US Government Users Restricted Rights -
D*Use, duplication or disclosure restricted
D*by GSA ADP Schedule Contract with IBM Corp.
D*
D*Description: Contains the Generic Record format headers
D*          for the user space.
D*
D*Header Files Included: none.
D*
D*Macros List: none.
D*
D*Structure List: Qus_Generic_Header_0100
D*             Qus_Generic_Header_0300
D*
D*Function Prototype List: none.
D*
D*Change Activity:
D*
D*CFD List:
D*
D*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
D*---- ------------ ----- ------ --------- ----------------------
```

```
D*$A0= D2862000     3D10  940213 LUPA:     New Include
D*End CFD List.
D*
D*Additional notes about the Change Activity
D*End Change Activity.
D*** END HEADER FILE SPECIFICATIONS *****************************
D*****************************************************************
D*Type Definition for the User Space Generic Header.
D*****************************************************************
DQUSH0100         DS                  BASED(SPCPTR)                    (4)
D*                                            Qus Generic Header 0100
D QUSUA                   1     64
D*                                            User Area
D QUSSGH                 65     68B 0
D*                                            Size Generic Header
D QUSSRL                 69     72
D*                                            Structure Release Level
D QUSFN                  73     80
D*                                            Format Name
D QUSAU                  81     90
D*                                            API Used
D QUSDTC                 91    103
D*                                            Date Time Created
D QUSIS                 104    104
D*                                            Information Status
D QUSSUS                105    108B 0
D*                                            Size User Space
D QUSOIP                109    112B 0
D*                                            Offset Input Parameter
D QUSSIP                113    116B 0
D*                                            Size Input Parameter
D QUSOHS                117    120B 0
D*                                            Offset Header Section
D QUSSHS                121    124B 0
D*                                            Size Header Section
D QUSOLD                125    128B 0
D*                                            Offset List Data
D QUSSLD                129    132B 0
D*                                            Size List Data
D QUSNBRLE              133    136B 0
D*                                            Number List Entries
D QUSSEE                137    140B 0
D*                                            Size Each Entry
D QUSSIDLE              141    144B 0
D*                                            CCSID List Ent
D QUSCID                145    146
D*                                            Country ID
D QUSLID                147    149
D*                                            Language ID
D QUSSLI                150    150
D*                                            Partial List Indicator
D QUSERVED00            151    192
D*                                            Reserved
D*****************************************************************
D*
D* The following QSYLOBJP include is copied into this program so
D* that it can be declared as BASED on LSTPTR, as shown at (5)
D* in the incorrect coding and at (6) in the correct coding.
D*
D*
D*****************************************************************
D*** START HEADER FILE SPECIFICATIONS ****************************
D*
D*Header File Name: H/QSYLOBJP
D*
D*Descriptive Name: List Objects That Adopt Owner Authority.
D*
```

```
D*
D*Description: Include header file for the QSYLOBJP API.
D*
D*Header Files Included: H/QSYLOBJP
D*                     H/QSY
D*
D*Macros List: None.
D*
D*Structure List: OBJP0100
D*             OBJP0200
D*             Qsy_OBJP_Header
D*
D*Function Prototype List: QSYLOBJP
D*
D*Change Activity:
D*
D*CFD List:
D*
D*FLAG REASON       LEVEL DATE   PGMR      CHANGE DESCRIPTION
D*---- ------------ ----- ------ --------- ----------------------
D*$A0= D2862000     3D10  931222 XZY0432:  New Include
D*
D*End CFD List.
D*
D*Additional notes about the Change Activity
D*End Change Activity.
D*** END HEADER FILE SPECIFICATIONS *****************************
D*****************************************************************
D*Prototype for calling Security API QSYLOBJP
D*****************************************************************
D QSYLOBJP        C                   'QSYLOBJP'
D*****************************************************************
D*Header structure for QSYLOBJP
D*****************************************************************
DQSYOBJPH         DS                  BASED(LSTPTR)                 (6)
D*                                            Qsy OBJP Header
D QSYUN00                 1     10
D*                                  User name
D QSYCV00                11     30
D*                                  Continuation Value
D*****************************************************************
D*Record structure for OBJP0100 format
D*****************************************************************
DQSY0100L02       DS                  BASED(LSTPTR)                 (6)
D*                                            Qsy OBJP0100 List
D  QSYNAME05              1     10
D  QSYBRARY05            11     20
D*                                  Qualified object name
D QSYOBJT12              21     30
D*                                  Object type
D QSYOBJIU               31     31
D*                                  Object in use
C*
C* Start of mainline
C*
C                   EXSR      INIT
C                   EXSR      PROCES
C                   EXSR      DONE
C*
C* Start of subroutines
C*
C*****************************************************************
C     PROCES        BEGSR
C*
C* This subroutine processes each entry returned by QSYLOBJP
C*
C*
```

```
C* Do until the list is complete
C*
C                   MOVE      QUSIS         LST_STATUS        1
C*
C     LST_STATUS    DOUEQ     'C'
C*
C* If valid information was returned
C*
C     QUSIS         IFEQ      'C'
C     QUSIS         OREQ      'P'
C*
C* and list entries were found
C*
C     QUSNBRLE      IFGT      0
C*
C* set LSTPTR to the first byte of the user space
C*
C                   EVAL      LSTPTR = SPCPTR
C*
C* increment LSTPTR to the first list entry
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSOLD + 1))
C*
C* and process all of the entries
C*
C                   DO        QUSNBRLE
C     QSYOBJT12     IFEQ      '*SQLPKG'
C*
C* Process *SQLPKG type                                          (2)
C*                                                              |
C                   ELSE                                        |
C     QSYOBJT12     IFEQ      '*PGM'                            |
C*                                                              |
C* Process *PGM type                                           |
C*                                                              |
C     QSYNAME05     DSPLY                                       |
C                   ELSE                                        |
C     QSYOBJT12     IFEQ      '*SRVPGM'                         |
C*                                                              |
C* Process *SRVPGM type                                        |
C*                                                              |
C                   ELSE                                        |
C*                                                             V
C*
C* Unknown type, log an error and exit from program (maybe..)
C*
C                   EXSR      DONE
C                   END
C                   END
C                   END
C*
C* after each entry, increment LSTPTR to the next entry
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSSEE + 1))
C                   END
C                   END
C*
C* When all entries in this user space have been processed, check
C* if more entries exist than can fit in one user space
C*
C     QUSIS         IFEQ      'P'
C*
C* by resetting LSTPTR to the start of the user space
C*
C                   EVAL      LSTPTR = SPCPTR
C*
C* and then incrementing LSTPTR to the input parameter header
```

```
C*
C                   EVAL      LSTPTR = %ADDR(ARR(QUSOIP + 1))
C*
C* If the continuation handle in the input parameter header is
C* blank, then set the list status to complete.
C*
C     QSYCV00       IFEQ      *BLANKS
C                   MOVE      'C'           LST_STATUS
C                   ELSE
C*
C* Else, call QSYLOBJP reusing the user space to get more
C* list entries
C*
C                   MOVE      QSYCV00       CONTIN_HDL
C                   EXSR      GETLST
C                   MOVE      QUSIS         LST_STATUS
C                   END
C                   END
C                   ELSE
C*
C* And if an unexpected status, log an error (not shown) and exit
C*
C                   EXSR      DONE
C                   END
C                   END
C                   ENDSR
C*****************************************************************
C     GETLST        BEGSR
C*
C* Call QSYLOBJP to generate a list
C* The continuation handle is set by the caller of this subroutine.
C*
C                   CALL      QSYLOBJP
C                   PARM                    SPC_NAME
C                   PARM      'OBJP0100'    MBR_LIST          8
C                   PARM      '*CURRENT'    USR_PRF          10
C                   PARM      '*ALL'        OBJ_TYPE         10
C                   PARM                    CONTIN_HDL       20
C                   PARM                    QUSEC
C*
C* Check for errors on QSYLOBJP
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QSYLOBJP'    APINAM           10
C                   EXSR      APIERR
C                   END
C                   ENDSR
C*****************************************************************
C     INIT          BEGSR
C*
C* One time initialization code for this program
C*
C* Set error code structure to not use exceptions
C*
C                   EVAL      QUSBPRV = %SIZE(QUSEC)
C*
C* Check to see if the user space was previously created in
C* QTEMP.  If it was, simply reuse it.
C*
C                   CALL      'QUSROBJD'
C                   PARM                    RCVVAR
C                   PARM                    RCVVARSIZ
C                   PARM      'OBJD0100'    ROBJD_FMT         8
C                   PARM                    SPC_NAME
C                   PARM      '*USRSPC'     OBJ_TYPE         10
C                   PARM                    QUSEC
C*
```

```
C* Check for errors on QUSROBJD
C*
C     QUSBAVL       IFGT      0
C*
C* If CPF9801, then user space was not found
C*
C     QUSEI         IFEQ      'CPF9801'
C*
C* So create a user space for the list generated by QSYLOBJP
C*
C                   CALL      'QUSCRTUS'
C                   PARM                SPC_NAME
C                   PARM      'QSYLOBJP  '  EXT_ATTR         10
C                   PARM                SPC_SIZE
C                   PARM                SPC_INIT
C                   PARM      '*ALL'        SPC_AUT          10
C                   PARM      *BLANKS       SPC_TEXT         50
C                   PARM      '*YES'        SPC_REPLAC       10
C                   PARM                QUSEC
C                   PARM      '*USER'       SPC_DOMAIN       10
C*
C* Check for errors on QUSCRTUS
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QUSCRTUS'    APINAM           10
C                   EXSR      APIERR
C                   END
C*
C* Else, an error occurred accessing the user space
C*
C                   ELSE
C                   MOVEL     'QUSROBJD'    APINAM           10
C                   EXSR      APIERR
C                   END
C                   END
C*
C* Set QSYLOBJP (using GETLST) to start a new list
C*
C                   MOVE      *BLANKS       CONTIN_HDL
C                   EXSR      GETLST
C*
C* Get a resolved pointer to the user space for performance
C*
C                   CALL      'QUSPTRUS'
C                   PARM                SPC_NAME
C                   PARM                SPCPTR
C                   PARM                QUSEC
C*
C* Check for errors on QUSPTRUS
C*
C     QUSBAVL       IFGT      0
C                   MOVEL     'QUSPTRUS'    APINAM           10
C                   EXSR      APIERR
C                   END
C                   ENDSR
C*****************************************************************
C     APIERR        BEGSR
C*
C* Log any error encountered, and exit the program
C*
C     APINAM        DSPLY                   QUSEI
C                   EXSR      DONE
C                   ENDSR
C*****************************************************************
C     DONE          BEGSR
C*
C* Exit the program
```

```
C*
C                   EVAL      *INLR = '1'
C                   RETURN
C                   ENDSR
```

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This API overview and concepts publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Printed in USA