AS/400 Advanced Series

System API Programming

Version 4

SC41-5800-00

AS/400 Advanced Series

System API Programming

Version 4

SC41-5800-00

Take Note!

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

First Edition (August 1997)

This edition applies to the licensed program IBM Operating System/400 (Program 5769-SS1), Version 4 Release 1 Modification 0, and to all subsequent releases and modifications until otherwise indicated in new editions.

Make sure that you are using the proper edition for the level of the product.

Order publications through your IBM representative or the IBM branch serving your locality. If you live in the United States, Puerto Rico, or Guam, you can order publications through the IBM Software Manufacturing Solutions at 800+879-2755. Publications are not stocked at the address given below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication. You can also mail your comments to the following address:

IBM Corporation
Attention Department 542
IDCLERK
3605 Highway 52 N
Rochester, MN 55901-7829 USA

or you can fax your comments to:

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

If you have access to Internet, you can send your comments electronically to IDCLERK@RCHVMW2.VNET.IBM.COM; IBMMAIL, to IBMMAIL(USIB56RZ).

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Notices

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, U.S.A.

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 the software interoperability coordinator. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Address your questions to:

IBM Corporation
Software Interoperability Coordinator
3605 Highway 52 N
Rochester, MN 55901-7829 USA

This publication could contain technical inaccuracies or typographical errors.

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

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

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

Programming Interface Information

This publication is intended to help experienced programmers create application programs. This publication documents General-Use Programming Interface and Associated Guidance Information provided by the Operating System/400 (OS/400) licensed program.

General-Use programming interfaces allow the customer to write programs that obtain the services of the OS/400 program. If you have a requirement for additional interfaces to the OS/400 program, contact IBM at 1-507-253-1055 (FAX 507-253-1571).

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

Advanced Function Printing
Advanced Peer-to-Peer Networking
AFP
Application System/400
APPN
AS/400
C/400
COBOL/400
DB2
FORTRAN/400
GDDM
IBM
Integrated Language Environment
NetView
OfficeVision
Operating System/2
Operating System/400
Operational Assistant
OS/400
PrintManager
RPG IV
RPG/400
SAA
SOM
SOMobjects
System/38
System/370
Systems Application Architecture
Ultimedia
400

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

C-bus is a trademark of Corollary, Inc.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About System API Programming (SC41-5800)

This book provides introductory, conceptual, and guidance information about how to use OS/400 application programming interfaces (APIs) with your application programs. It includes examples and discusses the following:

- Benefits of using APIs
- When to use APIs versus CL commands
- How to locate an API in the *System API Reference* book, SC41-5801
- Which high-level language (HLL) to use for APIs
- Various API environments
- Common information across object program model (OPM) APIs and Integrated Language Environment (ILE) APIs
- Characteristics and use of list APIs
- Differences between OPM and ILE APIs
- Machine interface (MI) programming
- Why you might want to make use of various OS/400 APIs
- Tips for common API programming errors

This book provides introductory and guidance information only; it does not contain API reference information. For reference information, see the *System API Reference* book.

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

Who Should Use This Book

This book is intended for experienced application programmers who are developing system-level and other OS/400 applications.

Prerequisite and Related Information

For information about other AS/400 publications (except Advanced 36), see either of the following:

- The *Publications Reference* book, SC41-5003, in the AS/400 Softcopy Library.
- The *AS/400 Information Directory*, a unique, multimedia interface to a searchable database that contains descriptions of titles available from IBM or from selected other publishers. The *AS/400 Information Directory* is shipped with the OS/400 operating system at no charge.

Information Available on the World Wide Web

More AS/400 information is available on the World Wide Web. You can access this information from the AS/400 home page, which is at the following uniform resource locator (URL) address:

`http://www.as400.ibm.com`

Select the Information Desk, and you will be able to access a variety of AS/400 information topics from that page.

Chapter 1. Application Programming Interface—Overview

Application programming interfaces (APIs) that are used on AS/400 business computing systems provide paths into system functions. APIs are intended for experienced application programmers who develop system-level applications and other AS/400 applications.

In the broadest sense, AS/400 **application programming interfaces (APIs)** are any formal interfaces that are intended to be used in the building of applications. These interfaces can include such functions as:

- Control language (CL) commands
- High-level language (HLL) instructions
- Machine Interface (MI) instructions
- Exit programs
- Callable Operating System/400 (OS/400) APIs, such as those discussed in the *System API Reference*, SC41-5801, and documentation for various licensed programs.

The APIs discussed in this book are those callable and bindable OS/400 APIs and OS/400 exit programs that are documented in the *System API Reference*.

Figure 1-1 on page 1-2 shows how APIs fit into the system structure.

API Compatibility

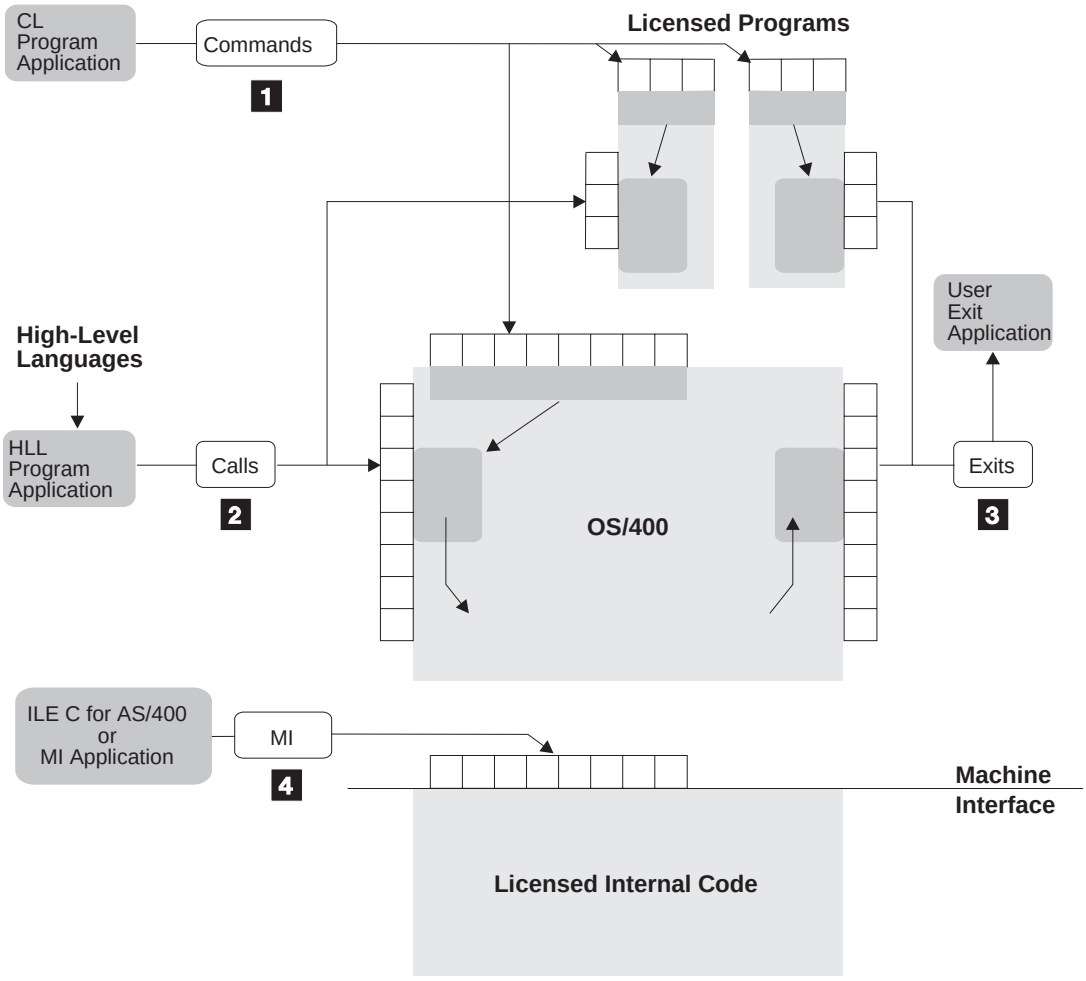
Original program model (OPM) APIs and Integrated Language Environment (ILE) APIs must be compatible from one release to the next. To ensure this compatibility, at least one of the following is true:

- Any additional parameters for existing OPM APIs are placed after the current parameters and are optional parameters. For example, the Move Program Message (QMHMOVPM) API has a group of required parameters and two groups of optional parameters.

  Note: ILE APIs cannot have additional parameters added to existing APIs.
- Any additional data structures are provided as a new format.
- Any new information for a data structure is added at the end of that format or replaces a field currently defined as reserved.

It is IBM's intention that the APIs will continue to work as they originally worked and any existing applications that use the APIs will continue to work without changes. However, significant architectural changes may necessitate incompatible changes.

Some API definitions (for example, the UNIX** type of API definitions) are established by industry standards organizations where the degree of compatibility is determined by the organizations.



1 CL commands
2 Callable programs
3 Exit programs
4 Machine Interface Instructions
(accessible through ILE C for AS/400 or MI programming languages)

RV3W217-1

Figure 1-1. How APIs Fit into the AS/400 Business Computing System Structure

Using APIs—Benefits

Although some CL commands and some OS/400 APIs perform the same basic functions, APIs often can provide additional performance improvements and access to functions. Benefits for using APIs include the following:

- APIs are slightly faster than the following:
  - Using the equivalent command or calling a CL program to call the command.
  - Coding a call to a command by using the Process Commands (QCAPCMD) API because the API is saved the overhead of processing a command. When you call an API, you do not have to go through the command analyzer.
  - Using the system function in the ILE C language that processes commands.

- Your application is more straightforward if you are coding in a programming language other than CL, which is not a fully defined language (it does not have the full capabilities of a high-level language). For example, you may have to code separate CL programs to perform specific functions.
- You can access system information and functions that are not available through CL commands.
- Data is often easier to work with when returned to you by an API.
- At times, you may need access to system functions at a lower level than what was initially provided on the AS/400 system. APIs and a set of documented machine interface (MI) instructions are available to allow the experienced programmer access to these system functions.

System APIs or CL Commands—When to Use Each

Before system APIs were offered on the AS/400, 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 AS/400 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 AS/400 system.

APIs are not provided as a replacement for CL commands, although in many cases there may 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. The CL command is intended to be entered either interactively or in a CL program, and the API is designed as a programming interface.

Some APIs have no equivalent CL command. 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.

Following are some of the types of APIs that perform actions; several examples of these APIs are discussed in more detail in later chapters of this book.

- List APIs, which return lists of information about something on the system. Refer to “List Object API—Examples” on page B-94 for an example of a list API in both ILE COBOL and ILE RPG.
- 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.

While many APIs are used alone, some can be used together to perform a task or function. The following is a list of a few functions:

- Defining, creating, distributing, and maintaining your own software products. See “Packaging Your Own Software Products” on page A-1 for an example of packaging a product similar to the way IBM packages products.
- Controlling systems and networks, which can include configuration, spooled files, network management, problem management, and so forth.
- Handling objects, which includes creating, changing, copying, deleting, moving, and renaming objects on the system.

APIs can also be categorized by the operating environment. For more information, refer to “API Environments” on page 2-4.

Related Information

Besides the OS/400 APIs that are documented in the *System API Reference*, other OS/400 APIs are documented in the following books:

- *Common Programming APIs Toolkit/400 Reference*, SC41-4802
- *CPI Communications Reference*, SC26-4399
- *DB2 for AS/400 Query Management Programming*, SC41-5703
- *GDDM Programming Guide*, SC41-0536
- *Machine Interface Functional Reference*, SC41-5810
- *PrintManager API Reference*, S544-3699
- *REXX/400 Programmer's Guide*, SC41-5728, and *REXX/400 Reference*, SC41-5729
- *Ultimedia System Facilities Programming*, SC41-4652

Many products on the AS/400 system also provide APIs. Refer to the product documentation for more information.

- Client Access
- OfficeVision for AS/400
- OSI File Services, OSI Message Services, and OSI Communications Subsystem
- System Manager for AS/400 and Managed System Services for OS/400
- TCP/IP Connectivity Utilities for AS/400

Chapter 2. Getting Started with APIs

You may find the information in this chapter helpful as you start to work with APIs; for example, locating the API that you want to use. It also tells you about information that you will need when using the APIs. Other topics covered are:

- Language selection considerations
- API environments
- API parameters
- Internal identifiers
- User spaces and receiver variables
- Continuation handles
- System and user domain concepts
- Exit programs
- QSYSINC library
- QUSRTOOL library
- User index considerations (recovering data)
- APIs and internal system objects
- Performance considerations

Locating the API to Use

If you are using the *System API Reference*, you can use the following methods to find an API:

- You can use the table of contents where the APIs are grouped by function. Within each chapter, the APIs are listed in alphabetical order.
- You can use the index where the APIs are listed in alphabetical order under the application programming interface (API) topic.
- You can also use the index where the APIs are listed under terms such as the following:
  - Job API
  - List API
  - Spooled file API
  - Socket network functions (in this case, the term *functions* means the same as APIs)

Except for APIs that are defined by formal standards organizations (for example, UNIX type or Systems Application Architecture (SAA)), OS/400 APIs start with the letter Q and are followed by two to three letters that make up an internal component identifier. The last part of the API name identifies the action or function of the API. Generally, the information after the component ID is an abbreviation of the verb that describes the function of the API. Figure 2-1 on page 2-2 contains all of the verbs that are either part of an API name or are implied verbs associated with an API name.

Figure 2-1. OPM and ILE API Verbs and Abbreviations

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

Figure 2-1. OPM and ILE API Verbs and Abbreviations

Verb	Abbreviation
Put	PUT, Put
PutGet	PutGet
Query	Q, QRY, Qry
Read	RD, Read, read
Receive	R, RCV, RECV
Register	RG, REG, R, Register
Release	RLS
Remove	RMV, Rmv, Remove, rm
Rename	RNM, rename
Report	Report
Resend	RSN
Reserve	Reserve
Restore	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

Note: Refer to “APIs for the ILE Common Execution Environment (CEE)” on page 2-5 for information about ILE CEE API names.

Selecting the High-Level Language To Use

You can use APIs with all the languages available on AS/400 business computing systems, except for the ILE APIs. ILE APIs that are implemented as service programs (*SRVPGM) can be accessed only by ILE languages. In some cases, a program (*PGM) interface is provided so that non-ILE languages can access the function.

Some APIs also require that particular data types and particular parameter passing conventions be used. Figure 2-2 shows the languages available on the AS/400 system and the data types that they provide.

Figure 2-2. Language Selection Considerations — Data Types

Language[1]	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[2]	X[2]	X		X	X
ILE C	X	X	X	X		X[9]	X	X	X	X
VisualAge C++ for OS/400	X	X	X	X		X[10]	X	X	X	X
CL		X[3]	X[3]	X		X		X[4]	X[4]	X
ILE CL	X[5]	X[3]	X[3]	X		X		X[4]	X[4]	X
COBOL	X	X	X	X	X	X		X	X	X[6]
ILE COBOL	X	X	X	X	X	X		X	X	X[6]
MI	X	X	X	X	X	X	X	X	X	X
Pascal (PRPQ 5799-FRJ)	X	X	X	X	X[7]	X[7]	X	X	X	X
PL/I (PRPQ 5799-FPJ)	X	X	X	X	X	X	X	X	X	X
REXX				X				X[4]	X[4]	X
RPG		X	X	X	X	X		X	X	X[8]
ILE RPG	X	X	X	X	X	X		X	X	X[8]

Notes:

1 You cannot develop Cross System Product (CSP) programs on an AS/400 system. However, you can develop CSP programs on a System/370 system and run them on your AS/400 system.
2 Refer to the CNVRT$ intrinsic function.
3 There is no direct support, but the %BIN function exists on the Change Variable (CHGVAR) CL command to convert to and from binary.
4 There is no direct support, but you can use the substring capability to simulate structures and arrays.
5 There is no direct support, but pointers passed to a CL program are preserved.
6 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.
7 There is no direct support, but you can use extended program model (EPM) conversion routines to convert to and from zoned and packed decimal.
8 RPG programs cannot monitor for specific messages, but these programs turn on an error indicator when a called program ends with an error. These programs can define an error handler to run when a program ends because of an error.
9 Packed decimal is implemented in ILE C with the decimal() data type.
10 Packed decimal is implemented in VisualAge C++ for OS/400 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.

Figure 2-3 on page 2-4 shows the languages available on the AS/400 system and the parameter support that they provide. For more information, see the reference manual for the specific programming language that you plan to use.

Figure 2-3. Language Selection Considerations — Call Conventions

Language[1]	Function Return Values[2]	Pass by Reference	Pass by Value
BASIC		X	
ILE C	X	X	X
VisualAge C++ for OS/400	X	X	X
CL		X	
ILE CL		X	
COBOL		X	[3]
ILE COBOL	X	X	X
MI		X	X
Pascal		X	
PL/I		X	
REXX		X	
RPG		X	
ILE RPG	X	X	X

Notes:

1 You cannot develop Cross System Product (CSP) programs on an AS/400 system. However, you can develop CSP programs on a System/370 and run them on your AS/400 system.
2 Return values are used by the UNIX-type APIs and the Dynamic Screen Manager (DSM) APIs.
3 COBOL provides a by-content phrase, but it does not have the same semantics as ILE C pass-by-value.

API Environments

OS/400 APIs exist in several operating environments on an AS/400 system. These environments are:

- Original program model (OPM)
- Integrated Language Environment (ILE)
- ILE Common Execution Environment (CEE)
- UNIX-type

APIs for the Original Program Model Environment

OPM APIs, the initial APIs on AS/400, use the following naming conventions:

- Start with the letter Q.
- Are followed by a 2- or 3-letter internal component identifier.
- Are limited to 8 characters.
- Must be uppercase.

Related Information

- *System API Reference*, SC41-5801
- *CL Reference*, SC41-5722
- Chapter 3, “Common Information across APIs—Basic (OPM) Example” on page 3-1

APIs for the Integrated Language Environment

The Integrated Language Environment (ILE) model is a set of tools and associated system support designed to enhance program development on an AS/400 system. Bindable ILE APIs are independent from the high-level languages. This can be useful when mixed languages are involved.

The ILE APIs provide functions such as:

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

ILE APIs use the following naming conventions:

- Start with the letter Q.
- Are followed by a 2- or 3-character internal component identifier.
- Can be up to 30 characters.
- Are case sensitive.

ILE service programs (*SRVPGM) use the following naming conventions:

- Start with the letter Q.
- Are followed by a 2- or 3-character internal component identifier.
- Are limited to 8 characters.
- Are uppercase.

Related Information

- Chapter 4, “Common Information across APIs—Advanced (ILE) Example” on page 4-1
- *ILE Concepts*, SC41-5606, for conceptual information about ILE
- Appropriate language guide or reference for information about the ILE languages
- *System API Reference*, SC41-5801

APIs for the ILE Common Execution Environment (CEE)

The ILE APIs with names beginning with CEE are based on the SAA language environment specifications. These APIs are intended to be consistent across the IBM SAA systems. CEE APIs with names beginning with CEE4 or CEES4 are specific to AS/400 business computing systems.

The ILE CEE APIs provide functions such as:

- 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 Information

- “Integrated Language Environment (ILE) CEE APIs” on page 8-7
- *ILE Concepts*, SC41-5606, for conceptual information about ILE
- *SAA CPI Language Environment Reference*, for information about the SAA Language Environment
- ILE CEE APIs in the *System API Reference*, SC41-5801

APIs for the UNIX Environment

The interfaces provided by sockets, the integrated file system, and the Common Programming APIs (CPA) Toolkit/400 are part of a continuing emphasis on supporting an open environment on the AS/400 system. The socket functions and integrated file system should ease the amount of effort required to move UNIX** applications to the AS/400 system.

The integrated file system is a function of OS/400 that supports stream input/output and storage management similar to personal computer and UNIX operating systems. It also provides an integrating structure over all information stored in AS/400.

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

Related Information

- “UNIX-Type APIs” on page 8-21
- *Common Programming APIs Toolkit/400 Reference*, SC41-4802
- *Integrated File System Introduction*, SC41-5711
- *Sockets Programming*, SC41-5422
- UNIX-type APIs in the *System API Reference*, SC41-5801

API Parameters

After you have found the API that you want to use, you need to code a call to an API and pass to the API the required set of parameters appropriate for that API. Parameters can be:

- Required, which means that you must enter all of the parameters in the specified order.
- Optional, which means that you must enter all or none of the parameters within the optional group. When you enter an optional group, you must also include all preceding optional groups.
- Omissible, which means that the group of parameters may have parameters that can be omitted. When these parameters are omitted, you must pass a null pointer.

For OPM and ILE APIs, the values for all parameters that identify objects on the system must be in *NAME (basic name) format, left-justified, 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 charac-

ters (A through Z, 0 through 9, $, #, @,), or _). The system uses an object name as is, and it does not change or check the object name before locating the object. This improves the performance of the API. An incorrect name usually results in an `Object not found` error.

Parameter Passing

With the exception of the UNIX-type APIs, the standard protocol is to pass a space pointer that points to the information that is being passed. (This is also referred to as pass-by-reference.) This is the convention automatically used by CL, RPG, and COBOL compilers. If you are using a language that supports pointers, you must ensure that these conventions are followed. Refer to the appropriate language documentation for instructions. Refer to “Selecting the High-Level Language To Use” on page 2-3 for a table that discusses the high-level languages.

In an OPM or ILE call, a **parameter** is an expression that represents a value that the calling application passes to the API specified in the call. HLL languages use the following methods for passing parameters:

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 operating system 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.

Figure 2-4 illustrates these parameter passing styles. Not all HLL languages support all styles.

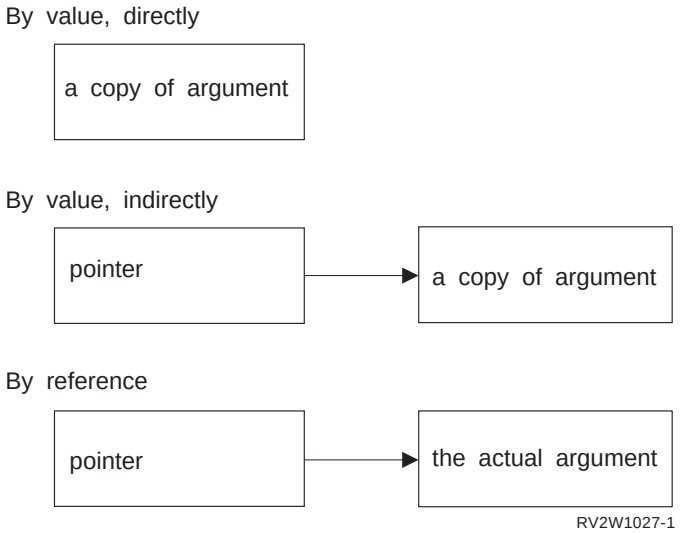


Figure 2-4. Methods for Passing Parameters

HLL 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 pro-

cedure passes parameters in the manner expected by the called API. The OPM or ILE HLL programmer's guides contain more information on 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 and COBOL support the passing of parameters by value (indirectly) and by reference.
- ILE RPG and RPG support the passing of parameters by reference.
- ILE CL and CL support the passing of parameters by reference.

Parameter Classification

Parameters can be classified into the following general categories:

- Input parameters: These parameters must be set to a value before calling the API because they 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 that object. Input parameters are not changed by the API.
- Output parameters: These parameters do not need to be set before calling the API because the API returns information to the application in these parameters. When a return to the application is successful and no errors have occurred, the application then accesses the information returned in output parameters.
- Input/Output parameters: These are parameters that 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 make up this parameter are output fields. The rules for input parameters and output parameters apply to the individual fields in the structure.

Error Code Parameter

The error code parameter is a variable-length structure that is a parameter for most APIs. (UNIX-type APIs and ILE CEE APIs do not use the error code structure.) The error code parameter controls how errors are returned to the application. The 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.

The structure for the error code parameter is discussed in the topic “Format of an Error Code Structure” on page 3-12. It contains information about the error code fields. Although the information is about an RPG example, the information is true for other APIs.

Note: The error code structure is provided in the QSYSINC library and is called QUSEC. Refer to Figure 2-5 on page 2-28 for a list of include files that are shipped in the QSYSINC library.

Receiving Error Conditions as Exceptions—Example

This example shows an application that receives error conditions as exceptions. It allocates an error code parameter that is a minimum of 4 bytes long 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	INPUT	OUTPUT
Bytes provided	0	0

Receiving the Error Code without the Exception Data—Example

This application example attempts to create an alert for message ID USR1234 in message file USRMSG in library QGPL. It receives the error condition in the error code parameter but does not receive any exception data. To do this, it 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.

When the application calls the Generate Alert (QALGENA) API, the alert table USRMSG is not found, and QALGENA returns exception CPF7B03. The error code parameter contains the data shown in the following table. In this example, 16 bytes are provided for data, but 36 are available. Twenty more bytes of data could be returned if the bytes-provided field were set to reflect a larger error code parameter (see “Receiving the Error Code with the Exception Data—Example”).

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

Receiving the Error Code with the Exception Data—Example

This application example attempts to create an alert for message ID USR1234 in message file USRMSG in library QGPL. It receives the error condition in the error code parameter and receives exception data as well. To do this, it allocates an error code parameter that is 116 bytes long—16 bytes for the bytes provided, bytes available, exception ID, and reserved fields, and 100 bytes for the 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.

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

Field	INPUT	OUTPUT
Bytes provided	116	116
Bytes available	Ignored	36

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

Using the Job Log to Diagnose API Errors

Sometimes an API may issue one or more messages that state that the API failed, and the messages may direct you to see the previously listed messages in the job log. If your application program needs to determine the cause of the error message, you can use the Receive Message (RCVMSG) command 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.

Receiving Error Messages from the Job Log—Example

To receive error messages from the job log using a CL program, specify the following:

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

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

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

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

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

             GOTO       CMDLBL(ALLDONE)

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

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

```
ALLDONE:    ENDPGM
```

As an alternative to using the job log, the following RPG program uses the error code structure to receive error messages:

```
H* ****************************************************************
H*                                                                *
H* MODULE: ERRCODE                                                *
H*                                                                *
H* LANGUAGE: RPG                                                  *
H*                                                                *
H* FUNCTION: THIS APPLICATION DEMONSTRATES THE USE OF THE         *
H*           ERROR CODE PARAMETER.                                *
H*                                                                *
H* APIs USED: QHFRTVAT, QHFCRTDR                                  *
H*                                                                *
H* ****************************************************************
H* ****************************************************************
H*                                                                *
H* THIS PROGRAM DOES SOME SIMPLE VERIFICATION ON AN HFS           *
H* DIRECTORY.  THE QHFRTVAT API IS USED TO VERIFY THE EXISTENCE   *
H* OF THE SPECIFIED DIRECTORY.  IF THE DIRECTORY DOES NOT EXIST,  *
H* AN ATTEMPT IS MADE TO CREATE THE DIRECTORY.                    *
H*                                                                *
H* THERE ARE THREE PARAMETERS TO THIS PROGRAM                     *
H*                                                                *
H*     1    INPUT   PATHNM - NAME OF DIRECTORY                    *
H*     2    INPUT   PATHLN - LENGTH OF PATHNM PARAMETER           *
H*     3    OUTPUT  SUCCES - INDICATES SUCCESS OR FAILURE         *
H*                           '0' SUCCESS                          *
H*                           '1' FAILURE                          *
H* ****************************************************************
ISUCCES      DS
I                                     B   1   40RETCOD
IPLENG       DS
I                                     B   1   40PATHLN
IBINS        DS
I                                     B   1   40RETDTA
I                                     B   5   80ATTRLN
IERROR       DS
I                                     B   1   40BYTPRV
I                                     B   5   80BYTAVA
I                                         9  15 ERRID
I                                        16  16 ERR###
I                                        17 272 INSDTA
C           *ENTRY    PLIST
C                     PARM           PATHNM 80
C                     PARM           PLENG
C                     PARM           SUCCES
C*
C* INITIALIZE BYTES PROVIDED AND THE ATTRIBUTE LENGTH VARIABLE
C*
C                     Z-ADD272       BYTPRV
C                     Z-ADD0         ATTRLN
C*
C* RETRIEVE DIRECTORY ENTRY ATTRIBUTES
C*
C                     CALL 'QHFRTVAT'
C                     PARM           PATHNM
C                     PARM           PATHLN
C                     PARM           ATTR    1
C                     PARM           ATTRLN
C                     PARM           ATTR
C                     PARM           ATTRLN
C                     PARM           RETDTA
C                     PARM           ERROR
C*
C* CHECK FOR DIRECTORY NOT FOUND OR FILE NOT FOUND ERRORS.
C* IF WE RECEIVE ONE OF THESE THIS IS THE INDICATION THAT
C* WE CAN TRY TO CREATE THE DIRECTORY.
C*
C           BYTAVA    IFEQ *ZERO
```

```
C                     Z-ADD0         RETCOD
C                     ELSE
C           'CPF1F02' IFEQ ERRID
C           'CPF1F22' OREQ ERRID
C* ************************************************************
C* THERE IS NO NEED TO REINITIALIZE THE ERROR CODE PARAMETER.
C* ONLY BYTES PROVIDED IS INPUT TO THE API; IT WILL RESET THE
C* ERROR CODE PARAMETER FOR US.  AFTER THE CALL TO QHFCRTDR,
C* BYTES AVAILABLE WILL EITHER BE 0 IF SUCCESSFUL OR NONZERO
C* IF THE CREATE FAILS.  WE DO NOT HAVE TO WORRY ABOUT THE
C* PREVIOUS ERROR CODE BEING LEFT IN THE ERROR CODE PARAMETER.
C* ************************************************************
C                     CALL 'QHFCRTDR'
C                     PARM           PATHNM
C                     PARM 20        PATHLN
C                     PARM           ATTR    1
C                     PARM 0         ATTRLN
C                     PARM           ERROR
C           BYTAVA    IFEQ *ZERO
C                     Z-ADD0         RETCOD
C                     ELSE
C                     Z-ADD1         RETCOD
C                     END
C*
C                     ELSE
C                     Z-ADD1         RETCOD
C                     END
C                     END
C*
C* PROGRAM END
C*
C                     SETON                     LR
```

Related Information

- "Retrieving the Hold Parameter (Exception Message)—OPM RPG Example" on page 3-6
- "Error Handling" on page 4-5
- The "API Error Reporting" topic in Chapter 2 of the *System API Reference*, SC41-5801

Internal Identifiers

You know of jobs, spooled files, and so forth, by their names. The system uses an ID that is associated with the name. The ID is assigned based on usage. Several of the APIs either require or allow you to use an internal ID. When you use an internal ID, it is generally faster because the system does not have to convert the external name to the internal ID.

A variety of terminology is used to identify internal IDs. For example:

- 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 on 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 would use the List Jobs (QUSLJOB) API, which provides the internal job ID for each job in the list. You could then use the

internal job ID to access a spooled file for a job with the Retrieve Spooled File Attributes (QUSRSPLA) API.

User Spaces

APIs that return information to a caller generally return the information in a **user space** (used by list APIs) or a **receiver variable** (used by retrieve APIs).[1]

The list APIs require a user space for returning 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.

Following are some of the advantages of using user spaces:

- User spaces can be automatically extendable.
- User spaces can be shared across jobs.
- User spaces can exist across IPLs.

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. The topic “Logic Flow of Processing a List of Entries” on page 2-15 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:

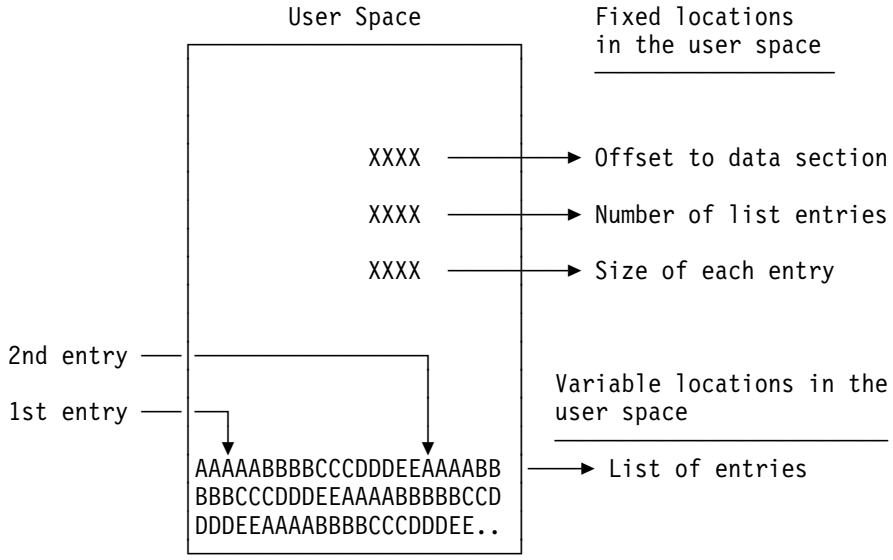
- If your language supports pointers, you can access or change the information directly. Figure 2-2 on page 2-3 describes each supported language and whether it supports pointers. Generally, pointer access is faster than API access.
- For languages that do not support pointers, you can use APIs to access 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. “Retrieving a File Description to a User Space—Examples” on page B-152 shows an example of pointer support.

[1] A **user space** is an object consisting of a collection of bytes that can be used for storing any user-defined information. A **receiver variable** is a program variable that is used as an output field to contain information that is returned from an API.

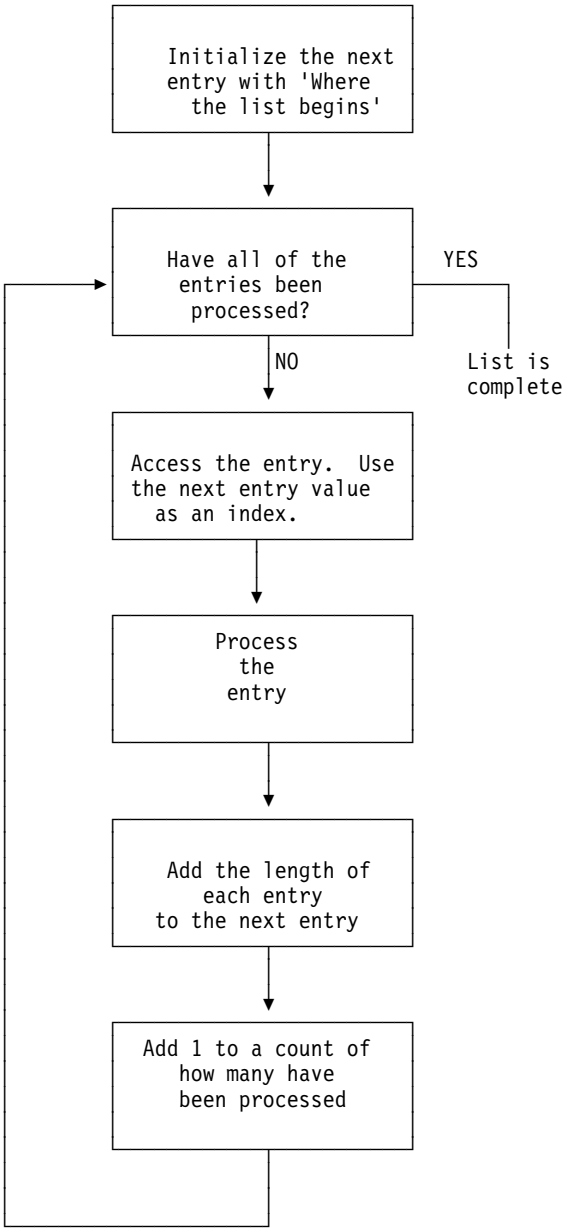
User Space Format—Example

Following is an example of the format of a user space. This example does not contain all of the fields in the fixed portion of a user space.



Logic Flow of Processing a List of Entries

When you process a list containing multiple entries, the logic flow looks as follows:



It is important from an upward compatibility viewpoint to use the offset, length of each entry, and the number of entries rather than hard coding the values in your program.

Related Information

- The User Space API chapter of the *System API Reference*, SC41-5801
- The “User Space Format for List APIs” topic in Chapter 2 of the *System API Reference*, SC41-5801

Manipulating a User Space with Pointers

Some languages, such as ILE C', VisualAge C++ for OS/400, ILE COBOL, ILE RPG, COBOL, Pascal, and PL/I, support pointers. Pointers allow you to manipulate information more rapidly from the user space. To use pointers with the OS/400 APIs, you should understand how to:

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

Synchronizing between Two or More Jobs

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

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

- 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 can occur while information is being updated. As a result, the data may not be accurate.

Using Offset Values with Pointers

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

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

See “Changing a User Space with an ILE RPG Program—Example” on page 2-20 for an example of this procedure.

Updating Usage Data

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

Manipulating a User Space without Pointers

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

Position Values

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

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

For examples of HLL programs that use positions, see “List Object API—OPM RPG Example” on page 5-4.

Lengths

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

Using Offset Values with the Change and Retrieve User Space APIs

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

Changing a User Space—Example

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

```
5763SS1 V3R1M0 940909            AS/400 DUMP       128747/ERICJ/ERICJS1    03/22/94 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-         3/22/93  11:02:56          SIZE-       0000400
OWNER-            ERICJ                      TYPE-       08   SUBTYPE-       01
ATTRIBUTES-               0800               ADDRESS-    01841400   0000
SPACE ATTRIBUTES-
 000000  00000080 00000060 1934E3C5 D4D7E2D7  C1C3C540 40404040 40404040 40404040 *    - TEMPSPACE                    *
 000020  40404040 40404040 E0000000 00000000  00000200 5C800000 00000000 00000000 *    \          *           *
 000040  00000000 00000000 00020002 6E000400  00000000 00000000 00000000 00000000 *      >                      *
SPACE-

 0000E0 5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C  5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C  **********************************
       LINES   000200     TO     0001FF   SAME AS ABOVE

.POINTERS-
    NONE
OIR DATA-
.TEXT-
 000000 E4A28599 40A29781 83854086 969940C3 88819587 8540E4A2 859940E2 97818385*User space for Change User Space*
 000020 40C5A781 94979385                                                 * Example                          *
.SERVICE-
 000000 40404040 40404040 40404040 40404040 40404040 40F14040 40404040 40404040*                   1              *
 000020 40404040 40404040 404040D9 F0F3D4F0 F0F0F9F0 F0F3F2F2 F1F1F0F2 F5F64040*         V3R1M00940322110256       *
 000040 40404040 40404040 40404040 40404040 40404040 40F14040 40404040 40404040*                                   *
 000060 40404040 40404040 40404040 40404040 40404040 40404040 00000000 00000000*                                   *
 000080 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000*                                   *
 0000A0 00000000 00000000                                                       *                                  *
 END OF DUMP
                                * * * * *  E N D  O F  L I S T I N G  * * * * *
```

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

RV3F089-0

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

```
5763SS1 V3R1M0 940909            AS/400 DUMP       128747/ERICJ/ERICJS1    03/22/94 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-        3/22/93  11:02:56          SIZE-       0000400
OWNER-           ERICJ                      TYPE-       08   SUBTYPE-       01
ATTRIBUTES-              0800               ADDRESS-    01841400   0000
SPACE ATTRIBUTES-
 000000  00000080 00000060 1934E3C5 D4D7E2D7  C1C3C540 40404040 40404040 40404040 *    - TEMPSPACE
 000020  40404040 40404040 E0000000 00000000  00000200 5C800000 00000000 00000000 *    \         *             *
 000040  00000000 00000000 00020002 6E000400  00000000 00000000 00000000 00000000 *      >                       *
SPACE-                                                                                                          *
 000000  C2898740 E2A39989 95874097 81848485  8440A689 A3884082 93819592 A2404040 *Big String padded with blanks  *
 000020  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040                                *
 000040  5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C  5C5C5C5C 5C5C5C5C 5C5C5C5C 5C5C5C5C ********************************
           LINES 000060   TO   0001FF  SAME AS ABOVE
.POINTERS-
    NONE
OIR DATA-
.TEXT-
 000000  E4A28599 40A29781 83854086 969940C3  88819587 8540E4A2 859940E2 97818385 *User space for Change User Space*
 000020  40C5A781 94979385                                                       * Example                       *
.SERVICE-
 000000  40404040 40404040 40404040 40404040  40404040 40F14040 40404040 40404040 *               1               *
 000020  40404040 40404040 404040D9 F0F3D4F0  F0F0F9F0 F0F3F2F2 F1F1F0F2 F5F64040 *      V3R1M000940322110256     *
 000040  40404040 40404040 40404040 40404040  40404040 40404040 40404040 40404040 *                               *
 000060  40404040 40404040 40404040 40404040  40404040 40404040 00000000 00000000 *                               *
 000080  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 *                               *
 0000A0  00000000 00000000                                                       *                               *
END OF DUMP

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

Area changed after using the Change User Space (QUSCHGUS)

RV3F088-0

Changing a User Space with an ILE RPG Program—Example

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

```
H*****************************************************************
H*
H* PROGRAM: CHANGUSPTR
H*
H* LANGUAGE: ILE RPG for OS/400
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
```

Changing a User Space with an OPM RPG Program—Example

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

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

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

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

When you are creating a list into a user space and the user space cannot hold all of the available information (the list is greater than 16MB 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).

Listing Database File Members with a CL Program—Example

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

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

Receiver Variables

Some APIs use receiver variables to place returned information. For example, instead of using a user space to return the information, the information is placed in a receiver variable. A retrieve API requires only addressability to storage of fixed size (typically a field or structure defined in your program), whereas a list API requires a user space because the amount of information returned by a list API may be large and not of a predictable size.

Retrieve APIs that return information to a receiver variable 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 may have one or many formats that give you the flexibility to choose the information that you need. Chapter 3, “Common Information across APIs—Basic (OPM) Example” on page 3-1 contains several examples of using receiver variables.

Some formats have variable-length fields, some only fixed-length fields, and yet others 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. For examples of repeating entry types and the various ways to move through receiver variable entries, see “Receiver Variables—Examples” on page 4-7.

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 will see the word displacement in the *Field* column of the API description.

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 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 could check the values for both the bytes available and bytes returned fields. If the bytes available field is greater than the bytes returned field, the API had more information to return than what would fit in the receiver variable. This could occur, over time, because the APIs that you use may be enhanced with new releases. The API may 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 was

returned on this API call. If both values are the same, the API returned 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 receiver variable parameter to the amount of storage allocated.
4. Pass the address of the storage allocated in step 2 by using pass by value (directly).

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 the user of the API to provide information to the API through the use of keys. Keys are API-specific values that inform the API that a certain function should be performed. Keys also are 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.

The keys are typically supplied to an API and are passed to the API using a variable-length record (there are some exceptions). 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. “Defining Byte Alignment” on page 9-22 shows examples of correctly and incorrectly defining byte alignment.

Some APIs that use variable-length records in addition to the List Job 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 later enhancing an API without affecting the user who chooses not to use the enhancements. For examples that use a keyed interface, see “Using Keys with List Spooled Files API—Example” on page 3-24 (OPM RPG) and “Keyed Interface—Example” on page 4-3 (ILE C).

User Space Alternative

Although a receiver variable is usually used for returning information from a retrieve API, sometimes a user space should be used instead. If the number of bytes of information being returned is not known or is large, a user space is preferred. You can create a user space so that it can automatically extend up to 16MB of storage to accommodate the information being retrieved.

The disadvantage of using a receiver variable when it may be too small for the amount of data being returned is that the API must be called again to receive the remaining data.

For an example of using a user space to return information from a retrieve API, see "Retrieving a File Description to a User Space—ILE C Example" on page A-11.

Related Information

- For a discussion of variable-length structures using ILE C, see "Variable-Length Structure—Example" on page 4-3.
- For an example using the Register Exit Point (QusRegisterExitPoint) API, see "Register Exit Point and Add Exit Program—ILE C Example" on page 4-9.
- For an example using the Change Object Description (QLICOBJD) API in RPG, see "Program for Packaging a Product—OPM RPG Example" on page A-3.

  The change object information parameter is defined as the COBJI field, and this field is later used by the QLICOBJD API.
- For a discussion of include files, see "APIs and the QSYSINC Library" on page 2-28.

Continuation Handle

Some APIs that return information offer a continuation handle. A **continuation handle** is a value that is passed between a high-level language program and an API. It is used to mark the last value put in either the receiver variable or the user space. When a call to an API is made and the API has more information to return than what could fit in the receiver variable or user space provided by the caller, the API returns a continuation handle. If a continuation handle is returned to the caller because there is more information to return, the caller can then call the API again and pass the continuation handle that was returned. The API continues to return information from the point that 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.

Using a Continuation Handle

To make use of a continuation handle, do the following:

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. Make use of 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 is returned. Keep all other parameters the same as the original API call.

For a program example that uses a continuation handle, see “Retrieve Exit Point and Exit Program Information—ILE C Example” on page 4-13.

Domain Concepts

All objects are assigned a domain attribute when they are created. A **domain** is a characteristic of an object that controls how programs can access the object. Once set, the domain remains in effect for the life of the object. The two 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, system domain objects can be accessed only by using the commands and callable APIs provided.

These 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 also can be in the user domain. Their contents cannot be manipulated directly by MI instructions.

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

The ability to create user domain objects on a system with a security level 40 or 50 is controlled by the allow user domain (QALWUSRDMN) system value. See the table in the description of the Create User Queue (QUSCRTUQ) API in the *System API Reference* for more information.

Note: On a system configured for C2[2] system security, QALWUSRDMN is set to QTEMP (only the QTEMP library can contain user-domain user objects).

For more information about C2 security, refer to the *Guide to Enabling C2 Security* book, SC41-0103.

Related Information

- "Using Data Queues versus User Queues" on page A-15
- Create User Index (QUSCRTUI), Create User Queue (QUSCRTUQ), and Create User Space (QUSCRTUS) APIs in the Object part of the *System API Reference*, SC41-5801
- Chapter 2 of the *Security – Reference*, SC41-5302

Exit Programs

Exit programs are called and given control by an application program or system program. They can be used to customize particular functions to your needs. An **exit program** is a program to which control is passed from a calling program.

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.

Prior to Version 3 Release 1, the exit program might have been represented as network attributes, system values, CL command parameters, or attributes of system objects. Also, in previous releases, all exit point providers had to supply their own means of registering and deregistering exit programs.

The **registration facility** provides a central point to store and retrieve information about OS/400 and non-OS/400 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.

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

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 Information

- For more information about the registration facility, see the Registration Facility part in the *System API Reference*, SC41-5801.
- For an example of adding an exit program to an exit point, see the topic "Register Exit Point and Add Exit Program—ILE C Example" on page 4-9.
- For an example of calling an exit program to retrieve information in the registration facility repository, see the topic "Retrieve Exit Point and Exit Program Information—ILE C Example" on page 4-13.

APIs and the QSYSINC Library

The QSYSINC (system include) library provides source include files shipped with the AS/400 system for OS/400 APIs. This optionally installable library is fully supported, which means that you can write APARs if you find a problem.

You can install this library by using the GO LICPGM functions of OS/400. Select the Install Licensed Programs option on the Work with Licensed Programs display and the OS/400 - 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 OPM API name or the ILE service program name. If the API has a callable and a bindable interface, an include file exists with both names.

The following table shows the API include files that are shipped with the QSYSINC library:

Figure 2-5 (Page 1 of 2). Include Files Shipped with the QSYSINC Library

Operating Environment	Language	File Name	Member Name (Header File)
OPM APIs	ILE C[1]	H	OPM API program name
	RPG	QRPGSRC	OPM API program name or OPM API program name with the letter E replacing the letter Q for members containing array definitions
	ILE RPG	QRPGLESRC	OPM API program name
	COBOL	QLBLSRC	OPM API name
	ILE COBOL	QCBLLESRC	OPM API program name

Figure 2-5 (Page 2 of 2). Include Files Shipped with the QSYSINC Library

Operating Environment	Language	File Name	Member Name (Header File)
ILE 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	ILE C	ARPA	Industry defined
	ILE C	H	Industry defined
	ILE C	NET	Industry defined
	ILE C	NETINET	Industry defined
	ILE C	SYS	Industry defined

Notes:

1 CEE ILE 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.

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

Include files for exit programs are shipped only if the exit program has a structure. The member names for these 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. Refer to the *System API Reference* for the actual member names for the exit programs.

All source physical files are shipped with read capabilities only; changes cannot be made to the QSYSINC library. All 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 coding in RPG or COBOL and need to define storage for variable length fields, you should copy the appropriate QSYSINC system include to a user source library. You can then customize the include file to your specific needs and use the customized member when you compile your application.

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

Related Information

- Chapter 2 in the *System API Reference*, SC41-5801

APIs and the QUSRTOOL Library

QUSRTOOL, an optionally installable library, has several tools that use APIs and include files (header files) for APIs. You can use the code as a model for your programming. As of Version 3 Release 6, source include files will be removed from this library. APARs against the offerings in QUSRTOOL are not allowed. There is no support for enhancements or fixes to the contents of the library other than a complete reshipment that occurs with each release. If you intend to use the tools, you should copy the members that you plan to use to your own source libraries.

User Index Considerations

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

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

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

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

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

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

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

You can log changes to a database file in a journal, and you can use the journal to apply or remove those changes later. You can also use the journal to audit who is using the database file. However, the system does not support the journaling of indexes. As a result, user applications should log entries in a journal to keep track of changes to the index, but you cannot update the index using apply and remove journal entry functions. For more information on journaling, see the Journal and Commit APIs in the *System API Reference* book, SC41-5801.

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.

APIs and Internal System Objects

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

Performance Considerations

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

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

The retrieve APIs, such as retrieve member description and retrieve spooled file attributes, have formats that are generally ordered from fastest performance to slowest performance. That is, the lower numbered formats run faster but retrieve less information, and the higher numbered formats run slower but retrieve more information. One exception is the Retrieve Job Information (QUSRJOBI) API where the order of the formats does not have anything to do with performance characteristics. For more information about the performance characteristics for the QUSRJOBI API formats, see the Retrieve Job Information (QUSRJOBI) API in the Work Management part of the *System API Reference* book.

Chapter 3. Common Information across APIs—Basic (OPM) Example

Through the use of several examples, this chapter provides information about how to use original program model (OPM) APIs in your programs. The primary example programs in this chapter are also shown in additional languages in “Original Program Model (OPM) APIs—Examples” on page B-2.

In the examples, the following are focus items:

- Description of an API by section
- Fixed-length formats
- Variable-length fields as output
- Optional parameters

The Retrieve Job Description Information (QWDRJOBD) API is used as the foundation for the examples in this chapter. The QWDRJOBD API has been included in “Retrieve Job Description Information API—Example” on page 3-29 for your use, if you would like to refer to it as you read this chapter.

For details on the OS/400 APIs, refer to the *System API Reference*, SC41-5801.

Original Program Model (OPM) API–Scenario

In this chapter, accessing information from a job description is used to demonstrate how to code APIs. While this may 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 Retrieve Job Description (RTVJOBD) command. The hold parameter determines whether the job is held on the job queue. Two values are supported:

***NO** The job is not held.
***YES** The job is held on the job queue.

Finding the API Name to Use

A first step in finding the correct API is to identify the part of the OS/400 program that is most closely related to the function in which you are interested.

If you want to access information from a job description, for example, you have to know that a job description object is considered part of the work management function. Next, you would turn to that chapter of the *System API Reference*.

Within each chapter of the *System API Reference*, the APIs are listed in alphabetical order by the spelled-out form of the API name. The API names contain verbs that are similar to the OS/400 licensed program: change, create, remove, and retrieve. Other verbs, such as list and set, you may not be familiar with. For more information on verbs, see Figure 2-1 on page 2-2. If you want to access information from a job description, the verb retrieve is a good place to start.

Retrieve functions are available for several work management objects, such as data areas, job descriptions, and job queues. Retrieve functions are also available for

nonobject information that represents data known to the system, such as job information or system status.

Description of an API

Most APIs have similar topic headings. The following lists the API topic headings, each with an overview and details on how to use the information.

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 value 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 to the API could be lost on the return from the call to the API.

In the fourth column of the Parameters box is the type of data defined for the parameter. CHAR(*) represents a data type that is not known, such as character, binary, and so on, or a length that is not known. Binary(*x*) represents *x* bytes of a binary value. CHAR(*x*) represents *x* bytes of character data. When calling the QWDRJOBD API, for example, there is 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 made up of several character and binary fields. For more information on format names, see “Format Name” on page 3-4.

RPG Call Statement—Parameter Example: In this example program, you must pass 5 parameters to use the API. For example, your RPG CALL statement might look like the following:

```
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 job description API. You will receive a data structure that contains information, and you will need to find the specific location within the data structure for where the HOLD information is stored.

Authorities and Locks

The Authorities and Locks topic lists all the authorities that you need to use the API. This topic also lists the locks that the API uses. To use an API, you must have the correct authority to the following:

- The API itself
- All the objects that the API uses
- 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 topic of an API lists all the parameters required for that API. You must use all of the parameters in the order that they are listed. None of the parameters may be left out.

The details of each parameter that must be used on the call to the QWDRJOBD API are described in “Required Parameter Group” on page 3-30.

Receiver Variable: A receiver variable is the name of the variable (QWDBH in the example RPG program in “Parameters” on page 3-2) where the information will be placed. 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 on the format, see the table in “JOBD0100 Format” on page 3-30.

You can see from the *Dec* (decimal offset) column of the JOBD0100 format table (**1** on page 3-31) that at least 390 bytes plus additional bytes (of unknown length) for the initial library list and the request data are returned. “Accessing a Field Value (Initial Library List)—OPM RPG Example” on page 3-19 describes how to determine the lengths of these fields. For now, you should focus on the fixed portion (390 bytes) of the format.

You have a choice of receiving 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 could specify that the receiver variable is 100 bytes (or any number greater than or equal to 86 bytes). It is not necessary to be precise when you specify 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, specify more length than the format has, and so on.

For example, assume that you decided to receive the fixed information, a length of 390 (**2** on page 3-7). 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 would usually use the length of the information that you want to receive. The length of 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 decided to declare the receiver variable to be 390 bytes in length. The length of receiver variable parameter will have a value of 390 assigned to it (**3** on page 3-7). 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 the example program in "RPG Call Statement—Parameter Example" on page 3-2, RCVLEN is the length of 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. The API does not care if the field is declared as a binary type. For example, some languages, like control language (CL), do not have a binary type. What is important is that the field is 4 bytes in length and that it contains the receiver length in binary.

If you write programs in CL, you need the %BIN function to convert a decimal value or variable to a character field that is declared as 4 bytes. If you write programs in RPG, you can declare a data structure that contains a 4-byte field of zero decimals and is defined as B for binary (**4** on page 3-7). Because the field is a binary type, RPG would make a binary value.

Format Name: A format name is a name that identifies what type of information you want returned in the receiver variable. Because this API has a single format name, JOBD0100, you would use the format name given (**5** on page 3-7) in the Retrieve Job Description Information API. 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, you would specify:

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

The special values of *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 name 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 if an exception occurred. Depending on your high-level language, you may not have a choice for which method you use. You may have to use the error-code data structure because some languages do not provide for escape messages.

In the example in "Retrieving the Hold Parameter (Exception Message)—OPM RPG Example" on page 3-6, the RPG program requests that exceptions be sent if any errors occur. To provide for this type of exception handling, a 4-byte binary field

with a value of zero (6 on page 3-7) must be passed. This indicates to the API that you want exception messages sent.

Optional Parameter Group

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

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

The Retrieve Job Description Information API has no optional parameter groups. The List Job (QUSLJOB) API is an example of an API with an optional parameter group.

Format

The Format topic describes a format name, which for the Retrieve Job Description (QWDRJOBD) API is JOBD0100. 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 Retrieve Job Description command) begins at decimal offset 76. For more information on this format, see “JOBD0100 Format” on page 3-30.

The fields in the format do not occur in any particular sequence. You have to scan the format to determine what you want.

This API has only a single format; other APIs may 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.

Field Descriptions

The Field Descriptions topic describes the fields found in the format. The contents of the format are presented in alphabetical sequence and not in the sequence of the fields defined in the format. In the Retrieve Job Description Information API example, 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.

Error Messages

The Error Messages topic lists error messages that can occur when you use the API. These are message IDs that normally exist in the QCPFMSG file. You may want to program for these messages regardless of the high-level language that you are using. If you need more detail about the messages, use the Display Message Description (DSPMSGD) command.

Extracting a Field from the Format

The format describes where the field that you want is located within the receiver variable. An offset is shown in both decimal and hexadecimal. Depending on which language you use, either offset may be helpful. For CL and RPG, you would normally use the decimal offset. With any offset, it is important to remember whether your language works with an offset from a base of 0 or a base of 1. The format tables in the *System API Reference* 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-justified in the field and the remaining positions are blank.

Most of the formats provide additional bytes for each field to allow for expansion, such as a new value for the hold on job queue field that would be 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 by QSYSINC. These variable-length fields must be defined by the user, as shown by **7** on page 3-20. For more information on the QSYSINC library, see "APIs and the QSYSINC Library" on page 2-28.

Retrieving the Hold Parameter (Exception Message)—OPM RPG Example

In the following program example, all the pieces have been put together with an RPG program that accesses the hold on job queue information from a job description. A message is sent for the value found. To make the RPG program more general purpose, two parameters for the job description (JOBD) name and library (JOBDL) name are passed to it **8** (refer to page 3-7). The program example, which is named JOBDAPI (this program name is also used in other examples in this chapter), does not handle errors. Any errors that are received are returned as exception messages.

```
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 2
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*              3                  4
I I           390                   B   1   40RCVLEN
I I           'JOBD0100'                5  12 FORMAT
I*             5
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           JOBDL  10
C*
C* Move the two parameters passed into LFNAM.
C*
C           JOBD      CAT  JOBDL     LFNAM  20 9
C* Error code bytes provided is set to 0
C*
C                     Z-ADD0         QUSBNB  6
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 by 3 on page 3-7.

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 5. (For program examples in other languages, see “Original Program Model (OPM) APIs—Examples” on page B-2.) The program generates the qualified name of the job description (JOBD) by concatenating the simple name and the library

qualifier 9 . 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).

Handling Error Conditions—OPM RPG Example

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 (10 on page 3-10). 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 (**11** on page 3-10). The message ID can be accessed from the program-status data structure. You would define this as follows:

```
I* Program status DS ( 12  on page 3-9)
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 (**13** on page 3-14).
- If you let exceptions occur, test the error indicator on the CALL operation (**10** on page 3-10).

Because you must test for some condition (one of the error messages in “Error Messages” on page 3-36), 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
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.

Retrieving the Hold Parameter (Error Code Structure)—OPM RPG Example

In the program example in "Retrieving the Hold Parameter (Exception Message)—OPM RPG Example," QUSBNB (**6** on page 3-7) 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 may 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" on page 3-36.
- 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) is:

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

The error code structure can be found in the QSYSINC library in the member QUSEC (see **14** on page 3-12). Which of the files you use depends on the language. For more information on the QSYSINC library, see "APIs and the QSYSINC Library" on page 2-28.

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** on page 3-7 in the JOBDAPI example on page 3-6.)

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 (**13** on page 3-14).

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:

```
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  14
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  15
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   13
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 include file QUSEC (**14** on page 3-12), and the program initializes the bytes provided field (QUSBNB) with a value of 16 (**15** on page 3-13). 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) (**13** on page 3-14). This contains the number of bytes of information about the error condition. The program is coded so 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 may 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” on page 3-12). 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
```

Handling Error Conditions—OPM RPG Example

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  (C) Copyright IBM Corp. 1994,1994
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  16

.
.
.

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 **16** 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 may be issued from the API. Appendix A of the *System API Reference* book contains an example of a diagnostic report program that uses the message handling APIs.

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.

Accessing the HOLD Attribute—OPM RPG Example

The following is the RPG code used to access the HOLD attribute. This is the same type of program as the RPG program examples in "Retrieving the Hold Parameter (Exception Message)—OPM RPG Example" on page 3-6 and "Retrieving the Hold Parameter (Error Code Structure)—OPM RPG Example" on page 3-11. The program, named JOBDAPI, prints the value of HOLD if it is found (**17** on page 3-19). If an error occurs, the program prints a line that contains the error message ID to a spooled file called QPRINT (**18** on page 3-19).

```
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  19
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 (**19** on page 3-17) 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  17
Failed. Error ID - CPF9801  18
```

Accessing a Field Value (Initial Library List)—OPM RPG Example

In this topic, the JOBDAPI program accesses a variable-length array. The variable-length array is the initial library list for the job description.

The discussion of the initial library list field in the job description format, "JOBD0100 Format" on page 3-30, indicates that the initial library list field is 11 bytes per entry, where each entry is a library name followed by a blank. Because the maximum number of libraries allowed on an initial library list is 25, this field is up to and including 275 bytes in length. 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 may begin in offset 390, but you should not rely on this. 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 two fields found in the format:

- Offset to the initial library list field (**19** on page 3-22 and on page 3-31).
- Number of libraries in the initial library list field (**20** on page 3-22 and on page 3-31).

If you use these field values in the format instead of hard coding an offset and a number of libraries, your program can work on any future release of an AS/400 business computing system, even if more job description attributes are defined in the format. This is an important upward compatibility approach that you will want to use 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.

```
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  (C) Copyright IBM Corp. 1994,1994
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  7
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  (C) Copyright IBM Corp. 1994,1994
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                                1000
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  19
I*                                         Offset Initial Lib List
I                                   B 365 3680QWDBH9  20
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            1000                 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.

Using Keys with List Spooled Files API—Example

This topic introduces a new 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 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 **1** .

```
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    5   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. See the “User Space Format for List APIs” topic in the book *System API Reference* for further information. 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. See “List Object API—Examples” on page B-94 for examples.

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, LSTSPL does the following:

- Initializes the COUNT variable to keep track of how many entries have been processed **17**
- Adds one to the base 0 offset (to the first entry in the list) as the QUSRTVUS API assumes base 1 positional values **18**
- Determines 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)

- Falls 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** .

Processing Lists That Contain Data Structures

Some APIs support a list where each entry in the list is itself 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.

For example, if another field is added to describe some 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 upwardly compatible while it retains the locations (relative to the start of a list entry) of the current fields.

Retrieve Job Description Information API—Example

The following API has been included as reference information. This API is used in many examples throughout this chapter and should not be used to code your programs. Refer to the *System API Reference*, SC41-5801, for the most recent version of this API.

Parameters

Required Parameter Group:

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

The Retrieve Job Description Information (QWDRJOBD) API retrieves information from a job description object and places it into a single variable in the calling program. The amount of information returned depends on the size of the variable.

The information returned is the same information returned by the Display Job Description (DSPJOBD) command.

Authorities and Locks

Job Description Object Authority *USE
Library Authority *USE

Required Parameter Group

Receiver variable

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

Length of receiver variable

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

Format name

INPUT; CHAR(8)
The format of the job description information to be returned. You can use this format:

JOBD0100 Basic job description information. For details, see "JOBD0100 Format."

Qualified job description name

INPUT; CHAR(20)
The name of the job description whose contents are to be retrieved. The first 10 characters contain the name of the job description, and the second 10 characters contain the name of the library where the job description is located. You can use these special values for the library name:

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

Error code

I/O; CHAR(*)
The structure in which to return error information. For the format of the structure, see "Error Code Parameter" in Chapter 2 of the *System API Reference*.

JOBD0100 Format

The following table describes the information that is returned in the receiver variable for the JOBD0100 format. For detailed descriptions of the fields, see "Field Descriptions" on page 3-32.

Offset			
Dec	**Hex**	**Type**	**Field**
0	0	BINARY(4)	Bytes returned
4	4	BINARY(4)	Bytes available

Offset			
Dec	Hex	Type	Field
8	8	CHAR(10)	Job description name
18	12	CHAR(10)	Job description library name
28	1C	CHAR(10)	User name
38	26	CHAR(8)	Job date
46	2E	CHAR(8)	Job switches
54	36	CHAR(10)	Job queue name
64	40	CHAR(10)	Job queue library name
74	4A	CHAR(2)	Job queue priority
76	4C	CHAR(10)	Hold on job queue
86	56	CHAR(10)	Output queue name
96	60	CHAR(10)	Output queue library name
106	6A	CHAR(2)	Output queue priority
108	6C	CHAR(10)	Printer device name
118	76	CHAR(30)	Print text
148	94	BINARY(4)	Syntax check severity
152	98	BINARY(4)	End severity
156	9C	BINARY(4)	Message logging severity
160	A0	CHAR(1)	Message logging level
161	A1	CHAR(10)	Message logging text
171	AB	CHAR(10)	Logging of CL programs
181	B5	CHAR(10)	Inquiry message reply
191	BF	CHAR(13)	Device recovery action
204	CC	CHAR(10)	Time-slice end pool
214	D6	CHAR(15)	Accounting code
229	E5	CHAR(80)	Routing data
309	135	CHAR(50)	Text description
359	167	CHAR(1)	Reserved
360	168	BINARY(4)	Offset to initial library list **19**
364	16C	BINARY(4)	Number of libraries in initial library list **20**
368	170	BINARY(4)	Offset to request data
372	174	BINARY(4)	Length of request data
376	178	BINARY(4)	Job message queue maximum size
380	17C	CHAR(10)	Job message queue full action
390 **1**	186	CHAR(*)	Reserved
*	*	ARRAY (*) of CHAR(11)	Initial library list
*	*	CHAR(*)	Request data

Field Descriptions

Accounting code. An identifier assigned to jobs that use this job description. This code is used to collect system resource use information. If the special value *USRPRF is specified, the accounting code used for jobs using this job description is obtained from the job's user profile.

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

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

Device recovery action. The action to take when an I/O error occurs for the interactive job's requesting program device. The possible values are:

**SYSVAL*
: The value in the system value QDEVRCYACN at the time the job is started is used as the device recovery action for this job description.

**MSG*
: Signals the I/O error message to the application and lets the application program perform error recovery.

**DSCMSG*
: Disconnects the job when an I/O error occurs. When the job reconnects, the system sends a message to the application program, indicating the job has reconnected and that the workstation device has recovered.

**DSCENDRQS*
: Disconnects the job when an I/O error occurs. When the job reconnects, the system sends the End Request (ENDRQS) command to return control to the previous request level.

**ENDJOB*
: Ends the job when an I/O error occurs. A message is sent to the job's log and to the history log (QHST). This message indicates that the job ended because of a device error.

**ENDJOBNOLIST*
: Ends the job when an I/O error occurs. There is no job log produced for the job. The system sends a message to the history log (QHST). This message indicates that the job ended because of a device error.

End severity. The message severity level of escape messages that can cause a batch job to end. The batch job ends when a request in the batch input stream sends an escape message, whose severity is equal to or greater than this value, to the request processing program. The possible values are from 0 through 99.

Hold on job queue. Whether jobs using this job description are put on the job queue in the hold condition. The possible values are *YES and *NO.

Initial library list. The initial library list that is used for jobs that use this job description. Only the libraries in the user portion of the library list are included.

Note: The data is an array of 11-byte entries, each entry consisting of a 10-byte library name left-justified with a blank pad at the end. The 11-byte entries can be easily used in CL commands. The number of libraries in the initial library list tells how many entries are contained in the array.

Inquiry message reply. How inquiry messages are answered for jobs that use this job description.

**RQD* The job requires an answer for any inquiry messages that occur while the job is running.

**DFT* The system uses the default message reply to answer any inquiry messages issued while the job is running. The default reply is either defined in the message description or is the default system reply.

**SYSRPYL* The system reply list is checked to see if there is an entry for an inquiry message issued while the job is running. If a match occurs, the system uses the reply value for that entry. If no entry exists for that message, the system uses an inquiry message.

Job date. The date that will be assigned to jobs using this job description when they are started. The possible values are:

**SYSVAL*
The value in the QDATE system value is used at the time the job is started.

job-date
The date to be used at the time the job is started. This date is in the format specified for the DATFMT job attribute.

Job description library name. The name of the library in which the job description resides.

Job description name. The name of the job description about which information is being returned.

Job message queue maximum size. The maximum size (in megabytes) of the job message queue. The possible values are:

0 The maximum size set by system value QJOBMSGMX at the time the job is started.

2–64 The maximum size of the job message queue in megabytes.

Job message queue full action. The action taken when the job message queue becomes full. The possible values are:

**SYSVAL* The value is specified by the system value QJOBMSGQFL.

**NOWRAP*
When the message queue becomes full, do not wrap. This action will cause the job to end.

**WRAP* When the message queue becomes full, wrap to the beginning and start filling again.

**PRTWRAP*
When the message queue becomes full, wrap the job queue and print the messages that are being overlaid.

Job queue library name. The library of the job queue into which batch jobs using this job description are placed.

Job queue name. The name of the job queue into which batch jobs using this job description are placed.

Job queue priority. The scheduling priority of each job that uses this job description. The highest priority is 1 and the lowest priority is 9.

Job switches. The initial settings for a group of eight job switches used by jobs that use this job description. These switches can be set or tested in a program and used to control a program's flow. The possible values are '0' (off) and '1' (on).

Length of request data. The length of all available request data, in bytes. If the receiver variable was not sufficiently large to hold all of the request data available, the amount of request data actually returned may be less than this value.

Logging of CL programs. Whether or not messages are logged for CL programs that are run. The possible values are *YES and *NO.

Message logging level. The type of information logged. Possible types are:

0 No messages are logged.

1 All messages sent to the job's external message queue with a severity greater than or equal to the message logging severity are logged.

2 The following information is logged:
- Level 1 information.
- Requests or commands from CL programs for which the system issues messages with a severity code greater than or equal to the logging severity.
- All messages associated with those requests or commands that have a severity code greater than or equal to the logging severity.

3 The following information is logged:
- Level 1 information.
- All requests or commands from CL programs.
- All messages associated with those requests or commands that have a severity greater than or equal to the logging severity.

4 The following information is logged:
- All requests or commands from CL programs.
- All messages with a severity code greater than or equal to the logging severity.

Message logging severity. The minimum severity level that causes error messages to be logged in the job log. The possible values are from 0 through 99.

Message logging text. The level of message text that is written in the job log or displayed to the user when an error message is created according to the logging level and logging severity. The possible values are:

**MSG*
Only the message is written to the job log.

**SECLVL*
Both the message and the message help for the error message are written to the job log.

**NOLIST*
If the job ends normally, there is no job log. If the job ends abnormally (the job end code is 20 or higher), there is a job log. The messages appearing in the job's log contain both the message and the message help.

Number of libraries in initial library list. The number of libraries in the user portion of the initial library list. Up to 25 libraries can be specified.

Offset to initial library list. The offset from the beginning of the structure to the start of the initial library list.

Offset to request data. The offset from the beginning of the structure to the start of the request data.

Output queue library name. The name of the library in which the output queue resides.

Output queue name. The name of the default output queue that is used for spooled output produced by jobs that use this job description.

**USRPRF* The output queue name for jobs using this job description is obtained from the user profile of the job at the time the job is started.
**DEV* The output queue with the same name as the printer device for this job description is used.
**WRKSTN* The output queue name is obtained from the device description from which this job is started.
output-queue-name
The name of the output queue for this job description.

Output queue priority. The output priority for spooled files that are produced by jobs using this job description. The highest priority is 1, and the lowest priority is 9.

Print text. The line of text (if any) that is printed at the bottom of each page of printed output for jobs using this job description. If the special value *SYSVAL is specified, the value in the system value QPRTTXT is used for jobs using this job description.

Printer device name. The name of the printer device or the source for the name of the printer device that is used for all spooled files created by jobs that use this job description.

**USRPRF*
The printer device name is obtained from the user profile of the job at the time the job is started.
**SYSVAL*
The value in the system value QPRTDEV at the time the job is started is used as the printer device name.
**WRKSTN*
The printer device name is obtained from the work station where the job was started.
printer-device-name
The name of the printer device that is used with this job description.

Request data. The request data that is placed as the last entry in the job's message queue for jobs that use this job description. The possible values are:

**NONE*
No request data is placed in the job's message queue.
**RTGDTA*
The data specified in the routing data parameter is placed as the last entry in the job's message queue.
request-data
The request data to use for jobs that use this job description.

Reserved. An ignored field.

Routing data. The routing data that is used with this job description to start jobs. The possible values are:

QCMDI
The default routing data QCMDI is used by the IBM-supplied interactive subsystem to route the job to the IBM-supplied control language processor QCMD in the QSYS library.

**RQSDTA*
Up to the first 80 characters of the request data specified in the request data field are used as the routing data for the job.

routing-data
The routing data to use for jobs that use this job description.

Syntax check severity. Whether requests placed on the job's message queue are checked for syntax as CL commands, and the message severity that causes a syntax error to end processing of a job. The possible values are:

-1 The request data is not checked for syntax as CL commands. This is equivalent to *NOCHK.

0–99 Specifies the lowest message severity that causes a running job to end. The request data is checked for syntax as CL commands, and, if a syntax error occurs that is greater than or equal to the error message severity specified here, the running of the job that contains the erroneous command is suppressed.

Text description. The user text, if any, used to briefly describe the job description.

Time-slice end pool. Whether interactive jobs using this job description should be moved to another main storage pool when they reach time-slice end. The possible values are:

**SYSVAL* The system value is used.
**NONE* The job is not moved when it reaches time-slice end.
**BASE* The job is moved to the base pool when it reaches time-slice end.

User name. The name of the user profile associated with this job description. If *RQD is specified, a user name is required to use the job description.

Error Messages

CPF1618 E Job description &1 in library &2 damaged.
CPF24B4 E Severe error while addressing parameter list.
CPF3CF1 E Error code parameter not valid.
CPF3CF2 E Error(s) occurred during running of &1 API.
CPF3C21 E Format name &1 is not valid.
CPF3C24 E Length of the receiver variable is not valid.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9804 E Object &2 in library &3 damaged.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.

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

Chapter 4. Common Information across APIs—Advanced (ILE) Example

Through the use of several examples, this chapter provides information about how to use more advanced or more complex API concepts in your programs. The example programs in this chapter use ILE APIs. These examples are also shown in additional languages in “Integrated Language Environment (ILE) APIs—Examples” on page B-47.

In the examples, the following are focus items:

- Header files
- Keyed interfaces
- Error handling
- Receiver variables

For more information about the APIs used in this chapter, refer to the *System API Reference*, SC41-5801.

Integrated Language Environment (ILE) APIs—Introduction

OS/400 Integrated Language Environment (ILE) APIs are contained within service programs that the calling program binds to. In addition, some OS/400 ILE 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 OS/400 ILE 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 contain OS/400 ILE APIs are in the QUSAPIBD binding directory. This binding directory is implicitly used by ILE compilers to resolve the OS/400 ILE API references; therefore, it is not necessary to explicitly name the service program or the API binding directory when creating programs that use OS/400 ILE APIs. If you provide your own APIs with the same name, make sure that you also provide your own binding directory or service program.

All OS/400 APIs (ILE and non-ILE) have a header file supplied by OS/400. These header files reside in the optionally installable library QSYSINC. The header files provide the prototypes for the API as well as 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 coding 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.

The example APIs in this chapter represent two general functions of APIs—change and retrieve. 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 an error did not occur when using APIs, the requested function completed successfully.

The presentation of the ILE APIs in the *System API Reference* is similar to the OPM APIs. For a general discussion of the API topics, see "Description of an API" on page 3-2.

Registration Facility Using ILE APIs—Concepts

The following examples illustrate the use of OS/400 ILE APIs. 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 may be 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.

The example programs are provided at the end of this chapter in their entirety (see "Registration Facility Using ILE APIs—Examples" on page 4-9). The example makes use of a continuation handle. Following are portions of the code to help illustrate concepts pertaining to the use of OS/400 ILE APIs. The following concepts include:

- Various types of header files
- The use of keyed interfaces
- Error handling and the error code parameter
- The use of receiver variables for returned information

Generic Header Files

This topic shows how to use a generic header file from the QSYSINC (system include) library in a program. For information about the QSYSINC header files, see "APIs and the QSYSINC Library" on page 2-28.

In addition to the traditional C-library header files (such as stdio and string), the API header file qusrgfa1.h is included in the following example. The qusrgfa1.h header file defines the functions exported from service program QUSRGFA1. This service program contains the APIs provided for manipulating the information in the repository. A second service program named QUSRGFA2 contains the ILE API for retrieving information from the registration facility repository. The header file qusec.h contains the definition for the error code structure that is used for the error code parameter. The following list shows the standard C header files (the first four includes) and a few AS/400-defined header files for APIs. This list does not show all the header files used in the example programs starting on page 4-9.

```
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <qusrgfa1.h>
#include <qusec.h>
```

Variable-Length Structure—Example

Many of the structures needed are provided by the QSYSINC (system include) library. However, any fields of a structure that are variable in length are not defined by QSYSINC and must be defined by the user. For example, in the qusec.h header file, the structure Qus_EC_t is defined as:

```
typedef struct Qus_EC {
     int  Bytes_Provided;
     int  Bytes_Available;
     char Exception_Id[7];
     char Reserved;
   /*char Exception_Data[]; 1 */   /* Varying length field  */
} Qus_EC_t;
```

Because the Exception_Data field **1** is a varying-length field, it is shown as a comment. The following is a new error code structure, which defines the exception_data field **2**. It was created by using the structure that was defined in the qusec.h header file.

```
typedef struct {
     Qus_EC_t ec_fields;
     char     exception_data[100] 2 ;
} error_code_struct;
```

Within QSYSINC include files, all varying-length fields and arrays are in the form of comments so that you can control how much storage to allocate based on your specific requirements.

Keyed Interface—Example

This example shows how to set key values for a keyed interface.

Exit Program Attributes Parameter

The interface for the Add Exit Program API is a keyed interface. One of the parameters for the Add Exit Program API is the exit program attributes parameter. The exit program attributes are provided to the API by means of a variable-length record. Typically, APIs that use a variable-length record interface use either a 3- or 4-field record. The Add Exit Program API makes use of a 4-field variable-length record **4**. The exit program attributes parameter for the API is defined as follows:

3

Type	Field
BINARY(4)	Number of variable-length records

4

Type	Field
BINARY(4)	Length of variable-length record
BINARY(4)	Exit program attribute key
BINARY(4)	Length of data
CHAR(*)	Data

The number of variable-length records field **3** is the first 4 bytes, and this field tells the API how many variable-length records have been specified. The fields defined in **4** are repeated (contiguously) immediately following the number of variable-length records for each record that is sent to the API. The API gets to each variable-length record by using the length of variable-length record field to get to the next record, up to and including the number of records specified.

The variable-length record structures are defined in the qus.h header file (this C-language header file is included by the qusrgfa1.h header file). The 4-field variable-length record is defined as:

```
typedef _Packed struct Qus_Vlen_Rec_4 {
     int  Length_Vlen_Record;
     int  Control_Key;
     int  Length_Data;
   /*char Data[];*/       /* Varying length field */
} Qus_Vlen_Rec_4_t;
```

Because the data field is a varying-length field, it needs to be defined in a new structure.

Exit Program Attribute Keys

The Add Exit Program API has several exit program attributes that can be set. The following table shows the valid exit program attribute keys for the key field area of the variable-length record.

Key	Type	Field
1	CHAR(27)	Qualified message file name and message identifier for exit program description
2	CHAR(50)	Exit program text description
3	BINARY(4)	Exit program data CCSID
4	CHAR(1)	Replace

This example specifies only two attribute keys (replace_rec and CCSID_rec fields) and lets the remaining attribute keys be set by the API to the default value. When working with variable-length structures, each variable-length record must start on a 4-byte boundary alignment. (The 4-byte boundary alignment requirement is only true for the registration facility APIs, not all keyed APIs.) The following new structure becomes the exit program attributes parameter on the call to the Add Exit Program API:

```
typedef struct {
     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;
```

The num_rec field is set to the value of 2 because the example specifies two variable-length records. The replace_rec field contains the length of the variable-length record (value of 16), the key (value of 4), and the length of the data (value of 1). The replace field contains the data for the replace key. The Reserved field reserves 3 bytes to force the next record to start on a 4-byte boundary alignment. The 3 bytes that are reserved are counted as part of the length for the replace variable-length record. The next record then follows the first record.

Error Handling

Error handling with ILE APIs can be accomplished in two ways: use the error code parameter or have exceptions signaled by the API to your application program.

Error Handling through the Error Code Parameter

The error code parameter enables a user to have exceptions returned to the program through the use of the parameter instead of having the exceptions signaled. Some exceptions may be signaled to the caller regardless of the size of the error code parameter. These exceptions are usually from errors that occur with the error code parameter itself (that is, message CPF3CF1) or with one of the other parameters (that is, message CPF9872). In the latter case, message CPF9872 is always signaled and never returned through the error code parameter because the API is unable to verify the error code parameter before the exception occurs.

The caller of the API must initialize the error code parameter so that the bytes provided field is set to the size, in bytes, of the error code parameter. For example, the error_code_struct structure on page 4-3 sets the size at 116 bytes. To initialize the error code parameter, do the following:

1. Allocate storage for the error code parameter:

   ```
   error_code_struct error_code;
   ```

2. Initialize the bytes provided field to the number of bytes that were allocated for the parameter:

   ```
   error_code.ec_fields.Bytes_Provided=sizeof(error_code_struct);
   ```

If the bytes provided field is set to a value equal to or greater than 8, the caller wants all exceptions returned through the error code parameter. The API fills in all of the message information up to the size of the error code parameter.

Error Determination—Example: On the return from the call to the API, verify whether or not an error occurred. If an error occurred, the bytes available field is set to something other than zero. If the bytes available field is not zero, you can use the message ID and the message data to determine what the program should do next. To receive the message ID and data, you must provide enough storage on the error code parameter for the information.

In the following example, the bytes available field is checked to determine if an error occurred. In this case, if an error occurred, a message is printed, which states the message ID and that the call to the API failed.

```
  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);
  }
```

Message Data—Example: If your program needs to handle different exceptions in different ways, you may need to make use of both the message data and the message ID. The message data is returned in the same format as when you display the message description on an AS/400. For example, if message CPF3C1E is received, some of the message data is printed. You can see message data that is associated with the message by using the Display Message Description (DSPMSGD) command. The message data for message CPF3C1E is defined as:

BIN(4)	Parameter number
CHAR(256)	ILE entry point name

To receive all of the message data for this exception, the exception data field of the error code structure would need to be at least 260 bytes in size. The following example uses only the number of bytes shown for the parameter number of the exception in the message data; therefore, the exception data field only needs to be 4 bytes.

```
int parm_number;
char *temp_ptr;

  if (error_code.ec_fields.Bytes_Available != 0)
  {
     if (memcmp(error_code.ec_fields.Exception_Id,"CPF3C1E",7)==0)
     {
       printf("\nFAILED WITH CPF3C1E:");
       temp_ptr=&(error_code.exception_data);
       parm_number=*((int *)temp_ptr);
       printf("\n  Parameter number omitted:  %d",parm_number);
     }
     else
     {
       printf("ATTEMPT TO REGISTER EXIT POINT FAILED WITH EXCEPTION: %.7s",
               error_code.ec_fields.Exception_Id);
       exit(1);
     }
  }
```

Error Handling Signaled by API

The second means of exception handling is by having all exceptions signaled by the API to the calling program. To have all exceptions signaled by the API, set the bytes provided field of the error code structure to zero. Refer to the documentation of your specific programming language for information on exception handling.

Receiver Variables—Examples

As discussed in “Receiver Variables” on page 2-23, receiver variables are generally used by retrieve APIs to return information to a caller. This topic provides coding examples of repeating entry types and of the use of offsets to go from one entry to the next in the receiver variable.

Repeating Entry Type with Fixed-Length Fields—Example

In the following example, the EXTI0100 format is defined in the qusreg.h header file, which is included by the qusrgfa2.h header file in the QSYSINC library.

This format is of the repeating entry type with all fixed-length fields. The repeating portion of the format (Qus_EXTI0100_Entry_t) is repeated after the fixed portion. The fixed portion of the format (Qus_EXTI0100_t) is returned only once. To go from one entry to the next, you add the offset exit point entry field to the starting position of the receiver variable to get to the first entry. Add the length exit point entry field to the current position in the receiver variable to move to the subsequent entries.

```
typedef _Packed struct Qus_EXTI0100_Entry {
   char Exit_Point_Name[20];
   char Format_Name[8];
   int  Max_Exit_Programs;
   int  Number_Exit_Programs;
   char Allow_Deregistration;
   char Allow_Change_Control;
   char Registered_Exit_Point;
   char Prep_Name_Add_Pgm[10];
   char Prep_Lib_Add_Pgm[10];
   char Prep_Format_Add[8];
   char Prep_Name_Rmv_Pgm[10];
   char Prep_Lib_Rmv_Pgm[10];
   char Prep_Format_Rmv[8];
   char Prep_Name_Rtv_Info[10];
   char Prep_Lib_Rtv_Info[10];
   char Prep_Format_Rtv[8];
   char Desc_Indicator;
   char Desc_Msg_File[10];
   char Desc_Msg_Library[10];
   char Desc_Msg_Id[7];
   char Text_Description[50];
 /*char Reserved[];*/
} Qus_EXTI0100_Entry_t;

typedef _Packed struct Qus_EXTI0100 {
   int  Bytes_Returned;
   int  Bytes_Available;
   char Continue_Handle[16];
   int  Offset_Exit_Point_Entry;
   int  Number_Points_Returned;
   int  Length_Exit_Point_Entry;
 /*char Reserved[];*/
 /*Qus_EXTI0100_Entry_t Array[];*/
} Qus_EXTI0100_t;
```

Repeating Entry Type with Variable-Length Fields—Example

In this example, the EXTI0200 format is defined in the qusreg.h header file, which is included by the qusrgfa2.h header file in the QSYSINC library.

This format is of the repeating entry type with some variable-length fields. The repeating portion of the format (Qus_EXTI0200_Entry_t) is repeated for each entry returned, and the fixed portion of the format (Qus_EXTI0200_t) is returned only once. To go from one entry to the next, you add the offset program entry field to the starting position of the receiver variable to get to the first entry. Then add the offset next entry field to the starting position of the receiver variable to get to each subsequent entry. To get to the Prog_Data field, add the offset exit data field to the starting position of the receiver variable and use the length exit data field to determine the number of bytes of information in the Prog_Data field.

```
typedef _Packed struct Qus_EXTI0200_Entry {
   int  Offset_Next_Entry;
   char Exit_Point_Name[20];
   char Format_Name[8];
   char Registered_Exit_Pt;
   char Complete_Entry;
   char Reserved[2];
   int  Program_Number;
   char Program_Name[10];
   char Program_Library[10];
   int  Data_Ccsid;
   int  Offset_Exit_Data;
   int  Length_Exit_Data;
 /*char Reserved[];*/
 /*Qus_Program_Data_t Prog_Data;*/
} Qus_EXTI0200_Entry_t;

typedef _Packed struct Qus_EXTI0200 {
   int  Bytes_Returned;
   int  Bytes_Available;
   char Continue_Handle[16];
   int  Offset_Program_Entry;
   int  Number_Programs_Returned;
   int  Length_Program_Entry;
 /*char Reserved[];*/
 /*Qus_EXTI0200_Entry_t Array[];*/
} Qus_EXTI0200_t;
```

Offsets Type—Example: The following portion of code illustrates the use of the offsets to go from one entry to the next in the receiver variable:

```
 /*****************************************************************/
 /* 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 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;
}
```

Registration Facility Using ILE APIs—Examples

Following are the entire program listings for the ILE C programs discussed in the preceding topics. The programs perform the following tasks:

- Register an exit point and add an exit program
- Retrieve exit point and exit program information
- Remove an exit program and deregister an exit point

These example programs are also shown in additional languages in “Integrated Language Environment (ILE) APIs—Examples” on page B-47.

Register Exit Point and Add Exit Program—ILE C Example

The following program registers an exit point with the registration facility and adds an exit program to the exit point.

```
/*************************************************************/
/*  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 */
```

Retrieve Exit Point and Exit Program Information—ILE C Example

The following program retrieves information about exit points and exit programs. It then resolves to each exit program and calls the exit program.

The Retrieve Exit Information API returns a continuation handle when it has more information to return than what fits in the receiver variable. For more information about continuation handles, see “Continuation Handle” on page 2-25.

```
/*****************************************************************/
/*  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 */
```

Remove Exit Program and Deregister Exit Point—ILE C Example

The following program removes an exit program from an exit point and deregisters the exit point from the registration facility.

```
/*********************************************************************/
/*  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 */
```

Chapter 5. List APIs

This chapter contains an overview of list APIs, which are those APIs that return a list unique to a given API. The chapter discusses the characteristics of a list API and provides information that you should be aware of when you use list APIs.

The List Objects That Adopt Owner Authority (QSYLOBJP) API is referred to throughout this chapter. The QSYLOBJP API is included in “List Objects That Adopt Owner Authority API—Example” on page 5-12 if you need to refer to it as you read this chapter.

Characteristics of a List API

As discussed in “User Spaces” on page 2-13, list APIs return information to a user space. List APIs generally have a user space parameter that uses a general (or common) data structure. You must use the general data structure to get at the information placed in the user space by the list API.

General Data Structure

This topic describes some of the more important fields that comprise the general data structure. Figure 5-1 on page 5-2 shows the common fields that 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 may specify the date last used, include comments about the list, and so forth.

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 may differ for each API.

Some fields may be added to the generic header from release to release. Because fields may be added, you may 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 input parameter section is an offset to the start of the input parameter section. The input parameter section may contain a copy of the input parameters that you pass to the list API. The QSYLOBJP API's input parameter section is shown on page 5-14.

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” on page 5-3 discusses continuation handles further.

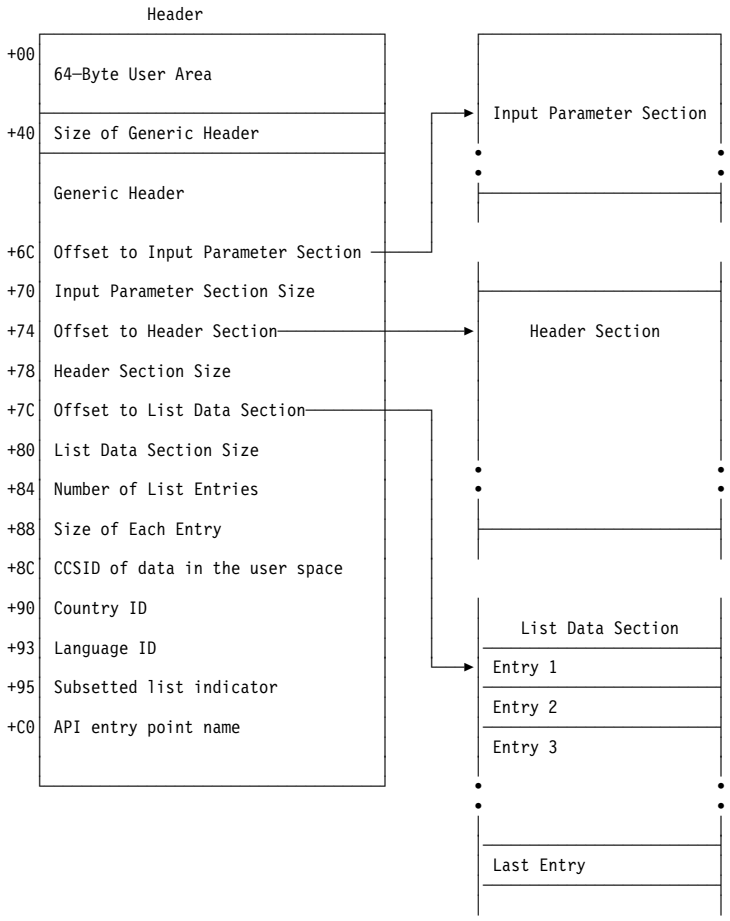


Figure 5-1. General Data Structure

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.

The QSYLOBJP API's header section is shown on page 5-14.

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” on page 5-14 and the “OBJP0200 Format” on page 5-14. This API has two possible formats.

For more information about formats and how to extract a field from a format, see “Format” on page 3-5 and “Extracting a Field from the Format” on page 3-5.

Other Fields of Generic Header

The field called structure's release and level is part of the generic header. This field tells the layout of the generic header. For an original program model (OPM) layout, this value should be 0100. For an Integrated Language Environment (ILE) model layout, 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 (see **1** on page 5-9). Possible values for this field follow:

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 (see **2** on page 5-14) to return the remaining information to the user space. The user then passes this value back to the API as an input parameter (see **3** on page 5-14) so that the API can locate the remaining information and place it in the user space.

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.

For more information about continuation handles and how to use them, see “Continuation Handle” on page 2-25.

Processing a List

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

1. Use the offset to list data section field (see 5 on page 5-9)
2. Look at the number of list entries field in the list (see 6 on page 5-10)
3. Add the size of each entry field to get to the start of the next entry (see 7 on page 5-9)

IBM may 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 would not process at the start of each entry.

The example program defines the size of each entry at 4 on page 5-9. For another example that shows the correct and incorrect way, see "Defining List Entry Format Lengths" on page 9-14.

List Object API—OPM RPG Example

The example program prints a report that shows all objects that adopt owner authority.

```
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: QRPGSRC/QUSEC
I*
I*Descriptive Name: Error Code Parameter.
I*
I*5763-SS1  (C) Copyright IBM Corp. 1994,1994
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
```

```
0                                       'Text: '
0                         QSYB6J
0*****************************************************************
0* Function:      ENDLST
0*
0* Description:   This function prints the end of listing print
0*                line and returns to the caller.
0*****************************************************************
0QSYSPRT E  106           ENDLST
0                                       '*** End of List'
```

List Objects That Adopt Owner Authority API—Example

Parameters

Required Parameter Group:

1	Qualified user space name	Input	Char(20)
2	Format name	Input	Char(8)
3	User	Input	Char(10)
4	Object type	Input	Char(10)
5	Continuation handle	Input	Char(20)
6	Error code	I/O	Char(*)

The List Objects That Adopt Owner Authority (QSYLOBJP) API puts a list of objects that adopt an object owner's authority into a user space.

This API provides information similar to that provided by the Display Program Adopt (DSPPGMADP) command.

Authorities and Locks

User Space Authority

*CHANGE

Authority to Library Containing User Space

*USE

User Profile Authority

*OBJMGT

Required Parameter Group

Qualified user space name

INPUT; CHAR(20)

The name of the existing user space to which the list of objects that adopt a user's authority is returned. The first 10 characters specify the user space name, and the second 10 characters specify the library. You can use these special values for the library name:

***CURLIB** The current library is used to locate the user space. If there is no current library, QGPL (general purpose library) is used.

***LIBL** The library list is used to locate the user space.

Format name

INPUT; CHAR(8)

The name of the format that returns information on the objects that adopt a user's authority.

You can specify these formats:

OBJP0100 Each entry contains the object name, library, type, and object in use indicator. For a detailed description of this format, see "OBJP0100 Format" on page 5-14.

OBJP0200 Each entry contains the same information as format OBJP0100 plus the object attribute and descriptive text. For a detailed description of this format, see "OBJP0200 Format" on page 5-14.

User name

INPUT; CHAR(10)

The user name for which the list of objects that adopt the user's authority is returned. You can specify the following special value:

***CURRENT** The list of objects that adopt the authority of the user currently running is returned. If *CURRENT is used, the name of the current user is returned in the list header section of the user space.

Object type

INPUT; CHAR(10)

The type of object for which the list of objects that adopt the user's authority is returned. You can specify only the following special values:

***ALL** Return entries for all object types that adopt authority.
***PGM** Return entries for programs that adopt authority.
***SQLPKG** Return entries for SQL packages that adopt authority.
***SRVPGM** Return entries for service programs that adopt authority.

Continuation handle

INPUT; CHAR(20)

The handle used to continue from a previous call to this API that resulted in partially complete information. You can determine if a previous call resulted in partially complete information by checking the Information Status variable in the generic user space header following the API call.

If the API is not attempting to continue from a previous call, this parameter must be set to blanks. Otherwise, a valid continuation value must be supplied. The value may be obtained from the list header section of the user space used in the previous call. When continuing, the first entry in the returned list is the entry that immediately follows the last entry returned in the previous call.

Error code

I/O; CHAR(*)

The structure in which to return error information. For the format of the structure, see "Error Code Parameter" in Chapter 2 of the *System API Reference*.

User Space Variables

The following tables describe the order and format of the data returned in the user space. For detailed descriptions of the fields in the tables, see “Field Descriptions” on page 5-15.

Input Parameter Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User space name specified
10	0A	CHAR(10)	User space library name specified
20	14	CHAR(8)	Format name
28	1C	CHAR(10)	User name specified
38	26	CHAR(10)	Object type
48	30	CHAR(20)	Continuation handle **3**

Header Section

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	User name
10	0A	CHAR(20)	Continuation handle **2**

OBJP0100 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Object in use

OBJP0200 Format

Offset		Type	Field
Dec	Hex		
0	0	CHAR(10)	Object name
10	0A	CHAR(10)	Library name
20	14	CHAR(10)	Object type
30	1E	CHAR(1)	Object in use
31	1F	CHAR(10)	Attribute
41	29	CHAR(50)	Text description

Field Descriptions

Attribute. The object attribute.

Continuation handle (header section). A continuation point for the API. This value is set based on the contents of the Information Status variable in the generic header for the user space. The following situations can occur:

- Information status–C. The information returned in the user space is valid and complete. No continuation is necessary and the continuation handle is set to blanks.
- Information status–P. The information returned in the user space is valid but incomplete. The user may call the API again, starting where the last call left off. The continuation handle contains a value which may be supplied as an input parameter in later calls.
- Information status–I. The information returned in the user space is not valid and incomplete. The content of the continuation handle is unpredictable.

Continuation handle (input section). Used to continue from a previous call to this API which resulted in partially complete information.

Format name. The name of the format used to return information on the objects that adopt authority.

Library name. The name of the library containing the user space or object.

Object name. The name of the object that adopts the user's authority.

Object in use. Whether the object is in use when the API tries to access it. If the object is in use, the API is not able to determine if the object adopts the user's authority. If the object is in use, this field is Y. If not, this field is N.

Object type.

- Input Section: The type of object for which the list of objects adopting the user's authority is returned.
- List Section: The type of object which adopts the user's authority.

Text description. The text description of the object.

User name. The name of the owner of the object.

User name specified. The name of the user for which the list of objects that adopt the user's authority is returned.

User space library name specified. The name of the library that contains the user space.

User space name specified. The name of the user space to which the list of objects that adopt the users authority is returned.

Error Messages

CPF22FD E Continuation handle not valid for API &1.
CPF2204 E User profile &1 not found.
CPF2213 E Not able to allocate user profile &1.
CPF2217 E Not authorized to user profile &1.
CPF3CF1 E Error code parameter not valid.
CPF3C21 E Format name &1 is not valid.
CPF3C31 E Object type &1 is not valid.
CPF811A E User space &4 in &9 damaged.
CPF9801 E Object &2 in library &3 not found.
CPF9802 E Not authorized to object &2 in &3.
CPF9803 E Cannot allocate object &2 in library &3.
CPF9807 E One or more libraries in library list deleted.
CPF9808 E Cannot allocate one or more libraries on library list.
CPF9810 E Library &1 not found.
CPF9820 E Not authorized to use library &1.
CPF9830 E Cannot assign library &1.
CPF9872 E Program or service program &1 in library &2 ended. Reason code &3.

Chapter 6. Original Program Model (OPM) and Integrated Language Environment (ILE) Differences

This chapter contains an overview of how Original Program Model (OPM) APIs and Integrated Language Environment (ILE) APIs differ from each other. The ILE APIs include the UNIX-type APIs and the ILE CEE APIs among others.

You must have the ILE language compiler on your system to develop applications that use any ILE APIs.

Contrasting OPM and ILE APIs

API Name

The maximum number of characters that an OPM API name can contain is 8, whereas the maximum number in an ILE API name is 30. ILE API names are case-sensitive.

Parameters

There are several types of parameters: required, optional, and omitted. The AS/400 OPM and ILE APIs show the parameters in a Parameter box, whereas the AS/400 UNIX-type APIs show them in a Syntax box. The ILE APIs include the service program name at the bottom of the box.

Optional Parameters: Some of the OPM APIs have optional parameters. The optional parameters form a group, and you must either include or exclude the entire group.

OPM APIs do not have omitted parameters.

Omitted Parameters: The ILE APIs may have parameters that can be omitted. When these parameters are omitted, you must pass a null pointer.

ILE APIs do not have optional parameters.

The required and optional parameters are discussed in more detail in “Required Parameter Group” on page 3-3 and “Optional Parameter Group” on page 3-5.

Error Conditions

The error code parameter is common to most of the OPM APIs, and 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 include the 7-character message identifier.

Some APIs use means other than the error code parameter for reporting error conditions, as follows:

- The ILE CEE APIs use feedback codes and conditions.
- The UNIX-type APIs use *errnos* and return values.
- The national language data conversion 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.

Pointers

Due to the greater availability of pointer support in ILE languages, there is a much greater use of pointers in ILE APIs. The use of pointers can provide a performance advantage.

Following are examples of an OPM API and an ILE API that do similar functions (log or report software errors). The ILE API example makes use of pointers, whereas the OPM API does not. Both programs log software errors by using first-failure data capture (FFDC).

Logging Software Error (OPM API without Pointers)—ILE C Example

This program calls the Log Software Error (QPDLOGER) API to perform FFDC. The program does not make use of pointers. The OPM program physically moves the data that is pointed to (shown at **1** on page 6-6), which slows down performance.

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

/*****************************************************************/
/*                    Miscellaneous Includes                     */
/*****************************************************************/
#include <qmhchgem.h>             /* from QSYSINC/H              */
#include <qpdloger.h>             /* from QSYSINC/H              */
#include <qusec.h>                /* from QSYSINC/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 */
```

Reporting Software Error (ILE API with Pointers)—ILE C Example

This program calls the Report Software Error (QpdReportSoftwareError) API to perform FFDC, and uses pointers. The ILE program sets a pointer (shown at **2** on page 6-10) to point to the same location as in the OPM program (shown at **1** on page 6-6).

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

/*****************************************************************/
/*                    Miscellaneous Includes                     */
/*****************************************************************/
#include <qmhchgem.h>             /* from QSYSINC/H              */
#include <qpdsrvpg.h>             /* from QSYSINC/H              */
#include <qusec.h>                /* from QSYSINC/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 */
```

Chapter 7. Machine Interface Programming

This chapter is for programmers interested in creating machine interface (MI) programs. While some MI instructions are discussed within the context of how to develop MI programs, this chapter makes no attempt to review the full range of MI instructions. The goal of this chapter is to provide a sufficient base of knowledge so that you can begin to use the MI language. After reading this chapter, you should be able to develop, create, run, and debug an AS/400 MI program. When reading this chapter, you will need access to the *System API Reference*, SC41-5801, and the *Machine Interface Functional Reference*, SC41-5810, because these two books are referred to extensively.

Machine Interface Instructions—Introduction

Programs and procedures are the two basic units of execution on the AS/400. 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 and its built-in MI support.

In the OPM environment, a program is comprised of two basic components: the object definition table (ODT) and an instruction stream. In the case of MI programs, the program is created using the Create Program (QPRCRTPG) API. This API is documented in the *System API Reference*.

The ODT is the means for defining all objects[1] 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 types of objects that can be declared are:

- 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 Functional Reference*.

[1] The term *objects* in this chapter refers to program data elements and not OS/400 object types such as a *FILE, *PGM, *USRPRF, and so on.

Within the source used to create a program, there is a type of statement called a directive. Directive statements are defined in the *System API Reference* in the section discussing 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 (*/).

Writing an MI Program—Example

This topic shows how to write a simple MI program that receives two packed-decimal parameters and returns the larger value through a third parameter. This program demonstrates how to do the following:

- Define an external entry point
- Define and access parameters
- Use conditional branching
- Assign a value to a scalar object
- End the program

Note: When reviewing this source code, unless noted otherwise, you can find all directive and DCL statements in the *System API Reference*; all other statements are in the *Machine Interface Functional Reference*. While this chapter attempts to discuss the intent of a statement, refer to the applicable reference book for specific details.

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

OS/400 programs typically pass parameters by reference as part of the high-level language (HLL) calling convention. Because OS/400 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 AS/400 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 in the *System API Reference*.

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 the *Machine Interface Functional Reference*. Pervasive instruction extenders such as branch (B) and target keywords (LO, HI, EQ, and so on) are defined in the *System API Reference* under "Instruction Statement," which is a subheading in the "Program Syntax" topic of the Create Program (QPRCRTPG) API.

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:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     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 a 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 documented in the *System API Reference*.

Note: The QPRCRTPG API assumes that the source statements presented to it are in code page 37. See the introduction to the *Machine Interface Functional Reference* for the specific code points required to build MI programs.

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” on page 7-11.)

Note: All non-MI source examples are provided in CL, because CL is the one language (other than REXX) that is standard on all AS/400 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.)

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. Test the MI01 program with the following CL01 CL program:

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*     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 on the AS/400 system. 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 AS/400 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:

```
5763SS1 V3R1M0 940909                              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                                                                       ;
5763SS1 V3R1M0 940909                              Generated Output                              08/08/94 09:46:36  Page     2
 MSGID    ODT   ODT Name                                    Semantics and ODT Syntax Diagnostics
5763SS1 V3R1M0 940909                              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 also can 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 7-6, 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 language 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, you should always refer to the *Machine Interface Functional Reference* to determine what specific exception IDs may 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” on page 7-27.

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.

The following 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
```

Creating an MI Version of CLCRTPG

The topics previous to this discuss how to create MI01 to be a reasonably complete 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 on page 7-13), 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 do the following:

- 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 appears as follows:

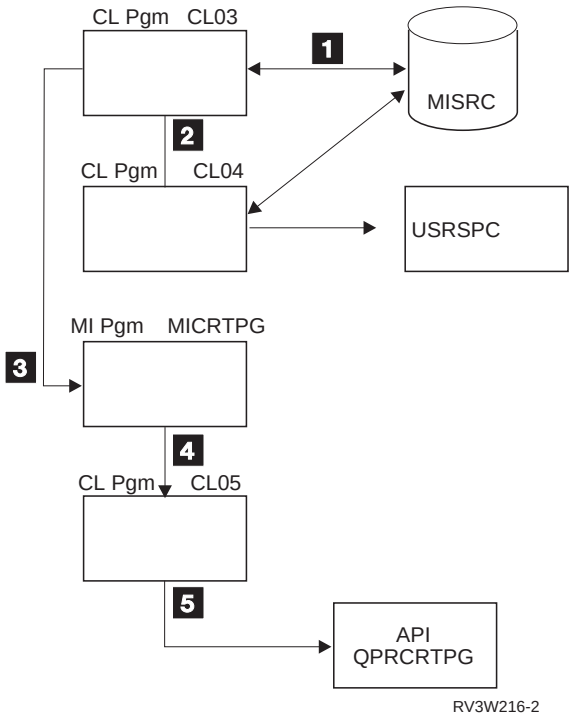


Figure 7-1. Program Flow for Creating the MICRTPG Program

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;
```

Understanding 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 an 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 (assigned from the source member name (MBR) data element). This user space is the one created in "Source for the CL03 Program" on page 7-13. If you are interested in the details of this structure, see the *Machine Interface Functional Reference* under RSLVSP. For other valid object types and subtypes, see *AS/400 Licensed Internal Code Diagnostic Aids – Volume 1*, LY44-5900, and *AS/400 Licensed Internal Code Diagnostic Aids – Volume 2*, LY44-5901.

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 *System API Reference* for this declare statement, 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 Functional Reference*.

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).

Enhanced Version of the MICRTPG Program

Now that the MICRTPG program provides addressability (through the *USRSPC as a parameter to the CL05 program) for as many as 64KB of input source, a new version of MICRTPG (named MICRTPG2) will incorporate 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 chapter.

The MICRTPG2 program demonstrates how to do the following:

- Receive a variable number of parameters
- Use static and automatic storage
- Create a space object
- Perform arithmetic operations

Understanding the MICRTPG2 Program (by Sections of Code)

Writing the program code for MICRTPG2:

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 MICRTPG2 create an automatically extendable space (it can automatically increase to as many as 16MB in size) using the Create Space (CRTS) instruction. Because the CRTS instruction requires a definition template, you need to define it (see the *Machine Interface Functional Reference* for details).

   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@, on the other hand, are being allocated from automatic (AUTO) storage. Many hours of debug time can be saved if you clearly understand how the AS/400 manages these two types of storage. For more information on the types of storage, see “AS/400 Program Storage” on page 7-36.

   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,” the instructions in steps 5, 6, and 7 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 invoked. 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 7-22, 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;
    ```

MICRTPG2 Complete Program—MI Code Example

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

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 may 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;
```

MICRTPG2 Complete Program (Enhanced)—MI Code Example

In its consolidated state, this is the new MICRTPG2 program:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     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;
```

MI Common Programming Techniques—Examples

With the completion of the MICRTPG2 program, the following example MI program demonstrates some additional programming techniques:

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*     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  */
/* OS/400 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;
```

AS/400 Program Storage

On AS/400 systems, two steps are needed to run a program: program activation and program invocation. **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 or not 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.

Chapter 8. Use of OS/400 APIs

This chapter discusses the various groups of OS/400 APIs and how they may be of use to you. Some APIs are discussed as a group, similar to the parts in the *System API Reference*, while others are discussed individually.

The API discussions in this chapter are presented in the same order as in the *System API Reference*.

Backup and Recovery APIs

Use of the backup and recovery APIs is described as follows:

- Operational-Assistant Backup APIs

  The APIs for Operational Assistant backup have been provided to give the users an interface into the Operational-Assistant backup setup functions without having to go through the normal displays. One possible use of these APIs would be to change the way the backup runs from one week to the next. For example, the user could write a CL program that could run on those weeks in which holidays occur to change the backup options to skip the backup for a certain day. This CL program could be submitted to run the week of a holiday to change the backup options to skip the backup on the holiday, and it could be submitted to run again after that week to set the options back to normal. The APIs could also be used to retrieve backup history about certain libraries, folders, and so forth, in order to better tailor the backups to get the most efficient backups.

- Retrieve Device Capabilities (QTARDCAP) API

  This API is useful for a tape management system. The API returns information about what capabilities your tape devices support.

- Save Object List (QSRSAVO) API

  This API is useful to get a level of granularity from your save operations that you cannot get by using the Save Object (SAVOBJ) command. The API allows you to associate specific object names with specific object types instead of saving the cross-product of all object names and object types entered on the command.

- List Save File (QSRLSAVF) API

  This API lists the contents of a save file into a user space. This is primarily for those who develop backup and recovery applications.

Client Support APIs

The client support APIs include the client software management and configuration APIs and exit programs.

The client software management and configuration section provides APIs to add, remove, refresh, and update client information on the AS/400 database with the information stored at the client. The client software management database formats are affected by these APIs. For information about these formats, see the chapter about client inventory management in the *Simple Network Management Protocol (SNMP) Support* book, SC41-5412. The exit programs notify you when these API functions have been completed.

With the exception of the Get Client Handle API, which is available only through the Integrated Language Environment (ILE), the client software management and configuration APIs are available as both the original program model (OPM) and ILE APIs.

- Add Client (QZCAADDC, QzcaAddClient) API

  To manage a client one must keep track of all clients on the network. This API provides a convenient way of keeping track of clients. By calling this API, a client is added to the database. If a client is SNMP-enabled and is set up in such a way so as to send traps to the managing AS/400, the client is automatically added to the AS/400 database. This API provides a way to add clients that may not be SNMP-enabled.

- Remove Client (QZCARMVC, QzcaRemoveClient) API

  This API provides a way to remove a client from the database for a client that is no longer required to be managed.

- Refresh Client (QZCAREFC, QzcaRefreshClientInfo) API

  If a client is SNMP-enabled, this API makes an attempt to get hardware and software information from this client. Hardware information is retrieved from the host resource management information base (MIB), and software information is retrieved from both the Desktop Management Interface (DMI) and the host resource MIB.

  This information is also retrieved automatically when traps are received on the managing AS/400 from its clients. Therefore, this API provides a way to force a “refresh” of client information to keep information current.

- Update Client Information (QZCAUPDC, QzcaUpdateClientInfo) API

  This API provides a way to update a few fields that are not updated with the Refresh Client API.

- Get Client Handle (QzcaGetClientHandle) API

  This API returns a handle, which is unique for every client known to AS/400. As mentioned earlier, the interface to this API is only provided through a service program.

Communications APIs

The user-defined communications APIs were created to provide users with the ability to develop their own high-level communications protocol with as little system interference as possible. While the system manages the lower-level protocol, the user develops the upper layers of a protocol in any high-level programming language supported by the AS/400. Several lower-level protocols are supported including X.25, Ethernet Version 2, IEEE Ethernet, token ring, and fiber distributed data interface (FDDI) (the selection of which is chosen by the user).

The APIs provide the ability to enable a link (that is, line, controller, and device), disable a link, establish inbound routing information by setting service access points (that is, filters), transmit and receive data, set timers, and query line descriptions.

The user-defined communications APIs are used primarily by users who have communications needs not normally associated with the other existing communications protocols, namely TCP/IP, SNA, or OSI. Applications that have been developed

using these APIs range from dedicated point-to-point file transfer to local-area-network client/server applications.

OptiConnect APIs

The OptiConnect APIs are used to move user data between two or more AS/400 systems that are connected by the OptiConnect fiber-optic bus. The OptiConnect APIs require that the OptiConnect hardware and software products have been installed on all of the systems that will be used for communications. A maximum of 32KB (where KB equals 1024 bytes) of data may be transferred in a single send or receive function.

Note: To use these APIs, you need the OptiConnect for OS/400 feature.

The OptiConnect APIs provide the following functions:

- Open and close an OptiConnect path
- Open and close an OptiConnect stream
- Send and receive a control message on an OptiConnect stream
- Send and receive a request or a message over an OptiConnect path
- Wait for a message on an OptiConnect stream

Configuration APIs

The configuration APIs can be used for the following functions:

- Change Configuration Description (QDCCCFGD) API

  This API allows a user to modify the values of parameters on existing AS/400 configuration descriptions. The primary purpose of this API is to allow support of new parameters or values as required, with their addition to the appropriate CL configuration command deferred to a later time. A primary user of this level of information would be user applications using new configuration capabilities not yet available through CL commands.

- List Configuration Descriptions (QDCLCFGD) API

  This API returns a list of configuration descriptions, based on a user-specified set of criteria.

- Retrieve Configuration Status (QDCRCFGS) API

  This API returns the current operational status of a specific configuration description on an AS/400.

Several APIs also provide feedback of information for any controller description, device description, and line description on an AS/400. The primary user of these APIs would be user applications performing system or network management functions on an AS/400. Separate formats are provided for each type of controller, device, and line.

Debugger APIs

The debugger APIs can be used for program debugging on the AS/400 system. The APIs are divided into separate sets of APIs, as follows:

- Integrated Language Environment (ILE) APIs
  - Source debugger APIs
  - Create view APIs
  - Dump Module Variable API

- Original program model (OPM) APIs
  - Retrieve Program Variable API

You can use these sets of source debugger APIs independently of each other or together as needed. The source debugger APIs can be used to write debuggers for the AS/400 system. The users of these APIs include:

- The source debugger that is shipped with the OS/400 licensed program. A **source debugger** is a tool for debugging Integrated Language Environment (ILE) programs by displaying a representation of their source code.
- Any other debugger that IBM or a business partner writes.

Debugger functions are designed to help you write and maintain your applications. You can run your programs in a special testing environment while closely observing and controlling the processing of these programs in the testing environment. You can write a debugger application that interacts with the debugger APIs, or you can use the debugger provided with the AS/400 system.

All debugger APIs must be called within the job in which the Start Debug (STRDBG) command is issued. The same program can be used at the same time in another job without being affected by the debugger functions set up.

To enable source-level debugging of ILE programs, view information must be stored with the compiled program. The ILE compilers use the create view APIs to create view information. This information is then available to source-level debugger applications through the source debugger APIs.

Dynamic Screen Manager APIs

The Dynamic Screen Manager (DSM) APIs are a set of screen I/O interfaces that provide a dynamic way to create and manage screens for the Integrated Language Environment (ILE) high-level languages. Because the DSM interfaces are bindable, they are accessible to ILE programs only.

The DSM APIs provide an alternative to the existing way of defining screen appearance outside a program by coding in data description specifications (DDS) or user interface manager (UIM), for example. Instead, programmers can use a series of calls to DSM within their programs to dynamically specify and control screen appearance for their applications. Unlike static definition methods, the DSM interfaces provide the flexibility needed for those applications requiring more dynamic screen control. The DSM support provided varies from low-level interfaces for direct screen manipulation to windowing support.

The DSM APIs fall into the following functional groups:

- **Low-level services**

  The low-level services APIs provide a direct interface to the 5250 data stream commands. These APIs are used to query and manipulate the state of the screen; to create, query, and manipulate input and command buffers used to interact with the screen; and to define fields and write data to the screen.

- **Window services**

  The window services APIs are used to create, delete, move, and resize windows, and to manage multiple windows during a session.

- **Session services**

  The session services APIs provide a general scrolling interface that can be used to create, query, and manipulate sessions, and to perform input and output operations to sessions.

Edit Function APIs

The edit function APIs are used to create and use edit masks. An **edit mask** is a byte string that tells the edit machine instruction or the Edit (QECEDT) API how to format a numeric value into a readable character string.

An edit mask can format a numeric value so that languages that cannot directly use machine instructions can now take advantage of this function. The edit mask was previously defined by the Edit Code (EDTCDE) and Edit Word (EDTWRD) keywords in DDS.

An **edit code** is a standard description of how a number should be formatted. There are many standard edit codes defined by the system. Users can define several edit codes the way they want with the use of the Create Edit Description (CRTEDTD) command.

An **edit word** is a user-defined description of how a number should be formatted. An edit word is usually used when one of the standard edit codes or user-defined edit codes is not sufficient for a particular situation.

The Convert Edit Code (QECCVTEC) API converts an edit code specification into an edit mask, and the Convert Edit Word (QECCVTEW) API converts an edit word specification into an edit mask. The resulting edit mask along with the value to be formatted are then passed to the Edit (QECEDT) API, which transforms the numeric from its internal format to a character form for displaying.

File APIs

File APIs provide complete and specific information about a file on a local or remote system. The file APIs include list, query, retrieve, and Structured Query Language (SQL) APIs. The list APIs generate lists of the following:

- Database file members
- How files and members are related to a specified database file
- Fields within a specified file record format name
- Record formats contained within a specified file

The Query (QQQQRY) API is useful for querying requests whereby you want a direct interface to OS/400 Query. The advantage of using the Query API is that you are not limited by the function of any particular product interface. Also, you can provide your own user interface and not be impeded by the extra code path associated with an extra layer such as SQL. It is currently used only by more expert programmers.

The Retrieve Display File (QDFRTVFD) API can be used to get complete and specific information about a display file. All the information that was in the data description specifications (DDS) is returned in the output of the QDFRTVFD API. By using the API, you can get specific field information for the display file.

The Retrieve File Override Information (QDMRTVFO) API retrieves the name of the file that will be referenced after file overrides have been applied to the file specified. A user program can retrieve the actual name of the file that will be used when the specified file is referenced.

SQL-related APIs process SQL extended dynamic statements in an SQL package object and call the DB2 SQL for OS/400 parser to check the syntax of an SQL statement.

Hardware Resource APIs

The hardware resource APIs allow you to work with hardware resources. A **hardware resource** is an addressable piece of hardware on the system. A hardware resource is known to the system by its **resource name**. A **resource entry** is the reference to the hardware resource in the hardware resource information, which can be thought of as a list of the hardware resources on the system.

The hardware resource APIs offer a more convenient means of accessing hardware resource information than its output file alternative. The output file may contain a great amount of information that is not needed by your application. The APIs, however, are a means of getting specific bits of information. For example, assume an application handles licensing of AS/400 information, and the application needs to know the type number of the system processor card. This type number represents the feature code of the system. To get this information previously, the application created an output file through a CL program, read each record of the file until it found the system processor card record, and then read the type field of this record. With the APIs, the type field can be returned directly.

Hierarchical File System (HFS) APIs

The HFS APIs provide applications with a single, consistent interface to the file systems registered with the hierarchical file system on your AS/400 system. The APIs automatically support the document library services (QDLS) file system and the optical file system (QOPT), and they can support user-written file systems also.

The HFS APIs allow you to work with nonrelational data stored in objects, such as directories and files in existing file systems. Using these APIs, you can perform such tasks as creating and deleting directories and files, reading from and writing to files, and changing the directory entry attributes of files and directories.

High-Level Language APIs

The high-level language part consists of the Application Development Manager APIs and the COBOL APIs.

Application Development Manager APIs

The Application Development Manager APIs allow a control language (CL) command such as the Build Part (BLDPART) command to determine, for example, the includes and external references that were used by certain processors (compilers or preprocessors) when processing a source member.

In terms of Application Development Manager, a part can be either a source member or an object, such as a file.

If you have an application that can use the information provided by the APIs, you can call these APIs from any high-level programming language. The Application

Development Manager feature does not need to be installed on your system for you to use these APIs.

The Application Development Manager APIs are:

- Get Space Status
- Read Build Information
- Set Space Status
- Write Build Information

The Get and Set Status APIs are used to query and initialize the build information space that is to contain the Application Development Manager information. The Write and Read Build Information APIs are used to write or read records of build information to and from the space.

COBOL APIs

These APIs let you control run units and error handling.

Integrated Language Environment (ILE) CEE APIs

The Integrated Language Environment (ILE) architecture on the OS/400 operating system provides a set of bindable APIs known as ILE CEE APIs. In some cases they provide additional function beyond that provided by a specific high-level language. For example, not all high-level languages (HLL) offer intrinsic means to manipulate dynamic storage. In these cases, you can supplement an HLL function by using appropriate ILE CEE APIs. If your HLL provides the same function as a particular ILE CEE API, use the HLL-specific one.

The ILE CEE APIs are useful for mixed-language applications because they are HLL independent. For example, if you use only condition management ILE CEE APIs with a mixed-language application, you will have uniform condition handling semantics for that application. This uniformity can make condition management easier than when using multiple HLL-specific condition handling models.

The ILE CEE APIs provide a wide-range of functions including:

Activation group and control flow management
Storage management
Condition management
Message services
Source debugger
Date and time manipulation
Math functions
Call management
Operational descriptor access

Naming Conventions of the ILE CEE APIs

Most ILE CEE APIs are available to any HLL that ILE supports. Naming conventions of the ILE CEE APIs are as follows:

- Bindable API names starting with CEE are intended to be consistent across the IBM SAA systems.
- Bindable API names starting with CEE4 are specific to AS/400.

Journal and Commit APIs

This section includes the journal and commitment control APIs.

Journal APIs

The journal APIs allow you to:

- Obtain information about some of the journal's attributes or the journal receiver's attributes
- Obtain journal information based on the journal identifier
- Send an entry to specified journal

Commitment Control APIs

The commitment control APIs allow you to:

- Add and remove your own resources to be used during AS/400 system commit or rollback processing
- Retrieve information about the commitment control environment
- Change commitment control options
- Put a commitment definition into rollback-required state

Message Handling APIs

On the AS/400, communications between programs, between jobs, between users, and between users and programs occurs through messages. The message handling APIs allow your application to work with these messages. The APIs consist of the following groups of functions:

- Send different types of messages to users and programs

  This would be done to communicate the status of an action that is about to occur or one that has been completed. You can also ask a question and wait for a response to it.

- Receive a message from a message queue

  This would be done to determine what action occurred, for example, to determine whether a function completed successfully (your program can continue) or failed.

- Handle errors that occur

  This is done to allow your application to tolerate errors that occur. For example, you can move and resend messages to another program for appropriate action. Or perhaps you expected the error, and you can just remove the message and continue.

- Return message or message queue information

  This is done to find attributes (for example, current delivery mode or severity) of a message queue or to return one or more messages on a message queue.

- Return message description or message file information

  This is done to find attributes of a message file or to return the actual description of a message in a message file.

Detailed information about these concepts and functions can be found in the *CL Programming* book, SC41-5721.

National Language Support APIs

These APIs provide the capability to retrieve cultural values, to convert sort sequence to different CCSIDs, to convert and truncate character strings, and to work with data that uses CCSIDs.

National Language Support APIs

This set of APIs gives you the capability to work with language IDs, sort sequence tables, and single- or double-byte character string data. The sort API allows a sort using a national language sorting sequence.

This set also includes an API that converts all characters to either uppercase or lowercase.

National Language Data Conversion APIs

This set of APIs provides the capability to convert character data from one CCSID to another.

Character Data Representation Architecture (CDRA) APIs

These APIs are part of the Character Data Representation Architecture that allows access to the system CCSID support.

Network Management APIs

Network management APIs give you the capability to manage one or more nodes from another node. The network management section consists of the following groups of APIs:

- Advanced Peer-to-Peer Networking (APPN) topology
- SNA/Management Services Transport
- Alert
- Node list
- Registered filter
- Change request management

Advanced Peer-to-Peer Networking (APPN) Topology Information APIs

APPN topology information APIs allow an application to obtain information about the current APPN topology, and to register and deregister for information about ongoing updates to the topology. The specific types of topology updates that an application may register to receiver follow:

- Network network node (*NN) updates
- Network virtual node (*VN) updates
- Local end node (*EN) updates
- Local virtual node (*VN) updates

APPN *network* topology identifies the following in an APPN subnetwork. (An **APPN subnetwork** consists of nodes having a common network ID and the links connecting those nodes.)

- All network nodes and virtual nodes in the subnetwork
- Transmission groups interconnecting network nodes and virtual nodes in the subnetwork
- Transmission groups from network nodes in the subnetwork to network nodes in adjacent subnetworks

The APPN *local* topology for an APPN node consists of the following:

- The local node
- Adjacent nodes (network nodes, end nodes, or virtual nodes to which the local node has a direct connection)
- Transmission groups from the local node to adjacent nodes

Both end nodes and network nodes can report local topology updates; however, network topology updates can be reported only on a network node system.

SNA/Management Services Transport (SNA/MS Transport) APIs

Systems Network Architecture Management Services Transport (SNA/MS Transport) functions are used to support the sending and receiving of management services data between systems in an SNA network. The network can include AS/400 systems, Operating System/2 and NetView licensed programs, and other platforms that support the SNA/MS architecture.

The SNA/MS functions provided on the AS/400 system include:

- The transport of network management data in APPN networks
- The maintenance of node relationships for network management

The APIs allow a network management application running on one system to send data to and receive data from a network management application running on another system in an APPN network. The APIs are a callable interface that allow the application to be notified about asynchronous events, such as incoming data, by way of a notification placed on a data queue.

Some examples of IBM applications that use SNA/MS Transport APIs are:

- Alerts
- Problem reporting
- Remote problem analysis
- Program temporary fix (PTF) ordering

In large networks, the number of sessions needed to support the various network management applications could become burdensome without session concentration. SNA/MS Transport APIs reduce the number of SNA LU 6.2 sessions that would normally be used to transmit data. This support multiplexes or transmits all of the network management data from all the applications in a network node domain (network node and attached end nodes) on a single session to applications in another domain.

This means that data transmitted from an end node is always sent to its network node server first. Then, the SNA/MS Transport support on the network node server routes the data to its proper destination.

Alert APIs

The alert APIs let your application create alerts, notify the OS/400 alert manager of alerts that need to be handled, and allow you to retrieve alerts and alert data. The generate and send APIs differ from ordinary AS/400 alert processing in that they let your application create an alert at any time without sending an alertable message to an alertable message queue. (An alertable message queue is a message queue that has been created or changed with the allow alerts (ALWALR) parameter speci-

fied as yes.) The retrieve API allows your application, in conjunction with alert filtering, to perform user-defined actions based on the contents of the alert.

Node List API

A **node list** is either or both of the following:

- A list of SNA nodes (network ID and control point name)
- Internet protocol nodes (internet address or host name)

You can use node lists for grouping systems by any criteria that you may need for your own applications. For example, the entries you put in a node list could be for systems at a certain hardware level. The List Node List Entries (QFVLSTNL) API is used to get the entries from the node list for use in your applications.

Registered Filter APIs

A **filter** is a function you can use to assign events into groups and to specify actions to take for each group. The registered filter APIs allow a product to register a filter with the operating system. The product can receive notification of events recorded in a data queue by using the Send to Data Queue (SNDDTAQ) action of the Work with Filter Action Entry (WRKFTRACNE) command.

A user filter is the filter defined by the network attributes for alert filtering and by the system value for problem log filtering. A user filter and a registered filter differ in their function and their notification record. There can only be one user filter active at one time for each type of filter, but there can be multiple registered filters active at one time. All actions are active for a user filter, but only the SNDDTAQ action is active for a registered filter.

A product can use registered filter APIs for the following purposes:

- To register multiple filters at the same time for each event type (alert or problem log)
- To deregister a filter when notifications from that filter are no longer necessary
- To retrieve all the filters that are registered

The event notification record for a registered filter differs from notification records for other types of filters. The registered notification contains a common header for all events, as well as specific information based on the type of event. The common header includes the name of the notification, a function type, a format, the filter name and library, the group name, and a timestamp. The specific information for the problem log includes the problem ID, the last event logged, and the timestamp for the last event.

Change Request Management APIs

This group of APIs can be used to add, remove, and list activities and to retrieve change request descriptions.

Object APIs

The object APIs consist of the following groups.

Data Queue APIs

Data queues are a type of system object that you can create, to which one high-level language (HLL) program can send data, and from which another HLL program can receive data. The receiving program can be waiting for the data, or can receive the data later.

The advantages of using data queues are:

- Using data queues frees a job from performing some work. If the job is an interactive job, the data queue APIs can provide better response time and decrease the size of the interactive program and its process activation group (PAG). This, in turn, can help overall system performance. For example, if several work station users enter a transaction that involves updating and adding to several files, the system can perform better if the interactive jobs submit the request for the transaction to a single batch processing job.
- Data queues are a fast means of asynchronous communication between two jobs. Using a data queue to send and receive data requires less system resource than using database files, message queues, or data areas to send and receive data.
- You can send to, receive from, and retrieve a description of a data queue in any HLL program. This is done by calling the Send to a Data Queue (QSNDDTAQ), Receive from Data Queue (QRCVDTAQ), Retrieve Data Queue Message (QMHRDQM), Clear Data Queue (QCLRDTAQ), and Retrieve Data Queue Description (QMHQRDQD) APIs.
- When receiving data from a data queue, you can set a time-out such that the job waits until an entry arrives on the data queue. This is different from using the EOFDLY parameter on the Override Database File (OVRDBF) command, which causes the job to be activated whenever the delay time ends.
- More than one job can receive data from the same data queue. This is an advantage in certain applications where the number of entries to be processed is greater than one job can handle within the desired performance restraints. For example, if several printers are available to print orders, several interactive jobs could send requests to a single data queue. A separate job for each printer could receive data from the data queue in first-in-first-out (FIFO), last-in-first-out (LIFO), or keyed-queue order.
- Data queues have the ability to attach a sender ID to each message being placed on the queue. The sender ID, an attribute of the data queue that is established when the queue is created, contains the qualified job name and current user profile.

Comparisons with Using Database Files as Queues: The following describes the differences between using data queues and database files:

- Data queues have been improved to communicate between active programs, not to store large volumes of data or large numbers of entries. For these purposes, use database files as queues.
- Data queues should not be used for long-term storage or indefinite retention of data. For this purpose, you should use database files.
- When using data queues, you should include abnormal end procedures in your programs to recover any entries not yet completely processed before the system is ended.

- It is good practice to periodically (such as once a day) delete and re-create a data queue at a safe point. Performance can be affected if too many entries exist without being removed. Re-creating the data queue periodically will return the data queue to its optimal size.

Similarities to Message Queues: Data queues are similar to message queues, in that programs can send data to a queue and that data can be received later by another program. However, more than one program can have a receive operation pending on a data queue at the same time, while only one program can have a receive operation pending on a message queue at the same time. (Only one program receives an entry from a data queue, even if more than one program is waiting.) Entries on a data queue are handled in either first-in first-out, last-in-first-out, or keyed-queue order. When an entry is received, it is removed from the queue.

User Queue APIs

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

You can use user queues to:

- Communicate between two processes asynchronously
- Store data in arrival sequence for later use
- Contain keyed messages
- Create a batch machine
- Permit better performance than the data queue interface

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

User Index APIs

The user index APIs allow you to:

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

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

You can use user indexes to:

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

- Sort data automatically based on the hexadecimal value of a key

For more information about user index considerations, refer to “User Index Considerations” on page 2-30. User index entries cannot contain a pointer. You can save and restore all the data in an index. You can also save and restore user indexes to another system.

User Space APIs

You can use these APIs to:

- Create and delete user spaces
- Change and retrieve the contents of user spaces
- Change and retrieve information about user spaces

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

You can use the user space APIs to:

- Create user spaces to be used by list APIs to generate lists of data.
- Store pointers.
- Store large amounts of data. You can create a user space as large as 16MB. You cannot create a data area larger than 2000 bytes.
- Save information in user space objects, and save and restore the object with the information in it using CL commands.
- Pass data from job to job or from system to system.

Object APIs

Use of the object APIs is described as follows:

- Change Library List (QLICHGLL) API

  This API provides the only way to change the product libraries in the library list. The only other way to change the product libraries is using the Create Command (CRTCMD) or the Create Menu (CRTMNU) command. You can also use this API to change the current library and the libraries in the user part of the library list similar to the Change Library List (CHGLIBL) command.

- Change Object Description (QLICOBJD) API

  Unlike the Change Object Description (CHGOBJD) command, this can be used on all external object types. This API supports changing more parts of the object descriptive information than are supported using the CHGOBJD command.

- Convert Type (QLICVTTP) API

  This API is the only supported way to convert a symbolic type to hexadecimal format and vice versa.

- List Objects (QUSLOBJ) API

  This API returns information similar to the Display Object Description (DSPOBJD) command. An advantage over the DSPOBJD command is that you can perform authority checking on the objects and libraries. You can get a

list of objects with only a certain status, which you cannot do with the DSPOBJD command.

- Rename Object (QLIRNMO) API

  This API combines the functions of the Rename Object (RNMOBJ) and the Move Object (MOVOBJ) commands. The API allows you to rename and move in one step, and replace the existing target.

- Retrieve Library Description (QLIRLIBD) API

  This API returns the number of objects in a library and the library size. Currently, the only other function that does this is the Display Library (DSPLIB) command with OUTPUT(*PRINT). Without this API, the user would have to generate a list of objects, using the Display Object Description (DSPOBJD) command, to an output file (or use the QUSLOBJ API), and then count the number of objects and total the size of the objects (and include the size of the *LIB object itself).

- Retrieve Object Description (QUSROBJD) API

  This API returns the same information as the Retrieve Object Description (RTVOBJD) command.

Office APIs

Descriptions of the office APIs follow:

- Display Directory Panels (QOKDSPDP) API

  This API can be called to change the system distribution directory interactively without using the OfficeVision administration interface.

- Display Directory X.400 Panels (QOKDSPX4) API

  This API adds an X.400 O/R name if one does not exist for a user. Also, the ability to display the O/R name is given. This API is for interactive use only and is useful if you want to add an X.400 O/R name to the directory interactively or to display an X.400 O/R name interactively.

- Search System Directory (QOKSCHD) API

  This API gives the ability to search any fields in the system distribution directory and return specified fields for each user that matches the search criteria. It also is used to query the actual fields that exist in the system distribution directory. It is most useful when the system distribution directory is used as a repository for information about users, and can be used in a program to query this information.

Other office APIs check spelling, and work with the document handling and document conversion exit programs, which are discussed under “Office Exit Programs” on page 8-16.

AnyMail/400 Mail Server Framework APIs

These APIs are used by those who are writing support for specific electronic mail (E-mail) server functions on the AS/400. An E-mail framework (the mail server framework or MSF), part of AnyMail/400, was added at Version 3 Release 1.

One API allows programs to create E-mail-related messages (MSF messages) that are then processed as part of this framework. The framework has a series of exit points that allow someone to plug in their own programs that are called when the

MSF framework processes its messages. These exit programs perform E-mail-related functions and are passed information about the MSF message. These exit programs can use two other APIs to retrieve or change part of the MSF message.

The remaining APIs are used to configure information to the MSF framework that it uses to tag and identify the parts of an MSF message. There are APIs to add, remove, and list these MSF-data-type definitions as part of MSF's configuration.

SNADS File Server APIs

You would use these APIs if you were dealing with AnyMail/400 mail server framework (MSF) messages that contained SNADS attachments. You would use the file server object read API if the MSF message was originated by OfficeVision or object distribution. You would use the create and write APIs if the MSF message was destined for an OfficeVision user or an object distribution user, or if you want to temporarily store attachments.

Office Exit Programs

Descriptions of the office exit programs follow:

- Directory Search exit program

  This exit program is provided for those who want the ability to search other files that contain user information. For example, when in OfficeVision and a user is on the Search System Directory display, if this exit program is provided, the F10 key is provided to call this exit program and as a result return the user ID and address.

- Directory Supplier exit program

  This exit program is provided for system distribution directory shadowing. It provides the ability to prevent add, change, and delete operations of select data from being supplied to a collector system. The advantages this provides is additional security to allow other systems on the network to see only the data you want them to see. It also gives you a way to provide partial replication through shadowing.

- Directory Verification exit program

  This exit program is provided to verify data that is being added, changed, and deleted in the system distribution directory. It is called for when data is directly being changed on the system and also when data is being changed through shadowing. This gives you the ability to verify that data is valid for specific fields (like telephone number format). It also gives you the added security of ensuring that the systems changing your data are authorized to do so. You could also use this exit program to provide partial replication for data that has already been sent from the other systems.

- Document Conversion exit program

  This exit program allows other document conversion programs to be called when a request is made for the OfficeVision program to process a document that is an unsupported type.

- Document Handling exit program

  This exit program allows other editors and applications to be called from the OfficeVision word processing and print functions.

- User Application Administration exit program

This exit program passes control to the application enabler where a registered alternate administration program will be called.

Operational Assistant APIs

Most functions on the AS/400 Operational Assistant menu can be accessed individually by calling APIs found in the QSYS library. The Operational Assistant APIs allow you to incorporate Operational Assistant functions into your application menus.

For information about the Operational-Assistant backup APIs, see “Backup and Recovery APIs” on page 8-1.

You can also tailor some of the Operational Assistant functions to your needs by using Operational Assistant exit programs.

Performance Collector APIs

The performance collector APIs allow applications to be developed that provide real-time performance monitoring capabilities. As with all APIs, this can be done without the overhead of databases or spooled files, and through a hardened interface. This is in contrast to the performance monitor, which collects much more information and retrieves its data much less frequently to database files. That is, in real-time, you have more control over collecting just the type of data you are interested in.

Print APIs

The print APIs consist of the following:

- Print APIs
- Spooled file APIs
- Exit programs

Print APIs

Print APIs can obtain information about or perform printing activities on the AS/400 system. Print APIs can:

- Retrieve output queue information such as status and number of entries on the queue.
- Retrieve information about specific printer writers.
- Transform data streams from one type to another.

The following discussion pertains to specific print APIs.

The AFP to ASCII Transform (QWPZTAFP) API converts an Advanced Function Printing data stream (AFPDS) into an ASCII printer data stream. The ASCII printer data streams supported are the printer control language Hewlett Packard** LaserJet**, Personal Printer Data Stream level 3 and 4 (IBM 4019, 4029), and PostScript** data stream. The API can be useful to anyone who wants to print AFP documents to lower-cost ASCII printers.

The Host Print Transform (QWPZHPTR) API converts an AFP data stream or an SNA-character-string (SCS) data stream to an ASCII data stream. The Host Print Transform API can be used as an alternative to printer emulation in the following situations to transform AFPDS and SCS data streams to ASCII data streams:

- Twinaxial ASCII printing
- LAN ASCII printing
- TCP/IP printing through the Send TCP/IP Spooled File command (known as LPR or line printer requester in UNIX TCP/IP)

The API provides a programming interface to the same transform.

The QWPZHPTR API goes along with another function, the Print Driver exit program. This exit program allows you to create your own print driver program. For example, the system provides a print driver to communicate to LAN-attached printers through the LexLink protocol. This print driver uses the host print transform to convert the SNA-character-string spooled files or AFP spooled files to ASCII before sending them to the printer.

With the QWPZHPTR API, you can write your own print driver (maybe to communicate over a protocol other than LexLink), but still use the system-provided transform (host print transform).

Spooled File APIs

Spooled file APIs obtain specific information about spooled files. For example, spooled file APIs can:

- Return a list of spooled files based on given selection criteria, such as a user or an output queue.
- Provide functions to access a specific spooled file from which the API can return the attributes and data of a spooled file or create a duplicate of a specific spooled file.

Spooled file APIs are useful in writing applications to clean up, save, and restore spooled files.

Problem Management APIs

The problem management APIs offer you the ability to write problem management solutions, improve serviceability, and manage your own applications. Problem management APIs deal directly with how the AS/400 handles problems today. Today, the problem log provides most of the operations necessary for problem management in a network environment. These APIs have several capabilities:

Filtering

A **filter** categorizes problem log entries into groups and performs operations on them accordingly.

The problem log applies the currently active filter to a problem log entry whenever a problem entry is created, changed, or deleted using system-provided interfaces.

The operations supported allow you to send application notification to a user data queue and assign the problem to a user. Your application can receive these notifications from the data queue using existing APIs.

Working with a Problem

Problem analysis is the process of finding the cause of a problem and identifying why the system is not working. Often this process identifies equipment or data communications functions as the source of the problem. The Work with Problem (QPDWRKPB) API allows you to perform problem analysis on local machine-detected problems in the problem log. The Work with Problem (QPDWRKPB) API prepares the problem in the problem log for reporting; it does not report the problem automatically.

Problem Log Entry APIs

The following set of problem management APIs are for designing problem log applications.

- Add Problem Log Entry (QsxAddProblemLogEntry)
- Change Problem Log Entry (QsxChangeProblemLogEntry)
- Create Problem Log Entry (QsxCreateProblemLogEntry)
- Delete Problem Log Entry (QsxDeleteProblemLogEntry)
- End Problem Log Services (QsxEndProblemLogServices)
- Retrieve Problem Log Entry (QsxRetrieveProblemLogEntry)
- Start Problem Log Services (QsxStartProblemLogServices)

Error Reporting APIs

The following set of problem management APIs log software problems.

- Log Software Error (QPDLOGER) reports a software problem and collects data needed for its resolution.
- Report Software Error (QpdReportSoftwareError) logs problems in the problem log and sends it to a service provider.

Program and CL Command APIs

You can use these APIs to do the following:

- Create programs
- List program or service program information
- Retrieve program or service program information
- Activate service programs
- Manipulate entries in the associated space of a program
- Handle compiler preprocessor-related tasks
- Resolve a pointer to an export
- Scan a string of characters for a pattern
- Execute a CL command or run a command from within an HLL or CL program

You can use the Create Program (QPRCRTPG) API to write your own assembler or compiler. When the assembler or compiler has created the machine interface template, this API is used to create the program from it.

Registration Facility APIs

The **registration facility** is a service that provides storage and retrieval operations for OS/400 and non-OS/400 exit points and exit programs. An **exit point** is a specific point in a system function or program where control may be passed to one or more specified exit programs. An **exit program** is a program to which control is passed from an exit point. This **registration facility repository** allows multiple programs to associate with a given system function or application function.

The registration facility APIs provide the capability to:

- Register and deregister exit points with the registration facility
- Add and remove exit programs to and from the repository
- Retrieve exit point and exit program information from the repository
- Designate the order in which exit programs should be called

An exit point can call one program, a fixed number of programs, or all programs associated with an exit point. The exit program number associated with each exit program should be used to determine the sequence in which the exit programs are run.

An exit point can be registered multiple times with the same exit point name; however, the combination of the exit point name and the exit point format name must be unique. Each exit program will be associated with a specific exit point and exit point format. The exit point format name can be used to indicate that a change occurred to the interface of the exit point. For example, this unique name (exit point and format) could be the result of a parameter change, version change, exit program data definition, and so forth. This unique name will facilitate having different exit programs run from different versions of a product for the same exit point name.

Security APIs

The OS/400 security APIs allow you to:

- Perform many of the security functions through a program interface. You can use APIs instead of CL commands.
- Combine many individual jobs into a single server or overhead job without compromising system security.

Network Security APIs

The OS/400 network security APIs provide a means that automatically logs you on to a server when you request a NetWare** function (for example, file or print). You can create authentication entries for each NetWare Directory Services** (NDS**) tree or NetWare 3.x server to which you are authorized. The entry identifies the tree or server, your name on that server, and (optionally) your password. When you request a NetWare function, the system attempts to start a connection to the server by using this data.

Note: To use these APIs, you need the Enhanced NetWare Integration for OS/400 feature.

Software Product APIs

The software products APIs were created to provide the user with the ability to package and manage their product in a manner similar to the way IBM licensed programs are managed. OS/400 commands along with these APIs allow you to work with and create program temporary fixes (PTFs), to package and distribute products, and to manage software licenses.

For more details on managing a product, see the *System Manager Use* book, SC41-5321. The System Manager for AS/400 licensed program can be used to facilitate managing your products.

The CD-ROM premastering APIs could be used if you currently produce distributed systems license option (DSLO) distribution tapes at a central site. If you would now like to distribute on CD-ROM rather than tape, you would use the CD-ROM premastering APIs.

You would use the Handle CD-ROM Premastering State (QlpHandleCdState or QLPCDRST) API to place your job into a CD-ROM premastering state. While in this state, any save operations performed will have information about the tape file sizes stored away for future use in generating the QDSETMAP file. In the case of a SAVSYS or option 40 (Create Distribution Tape) on the Work with Licensed Programs menu, special files are saved to tape so that the CD-ROM volumes produced can be used for installation.

When the save operations have been performed, you would then use the Generate CD-ROM Premastering Information (QlpGenerateCdPremasteringInfo or QLPCDINF) API. This API is used to analyze the tape file size information stored away during the previous API and produce a byte-stream file that contains information about which CD-ROM volumes these files will reside on. Information is also returned into a user space that is useful in producing a mastering control file that may be necessary when having the CD-ROMs mastered.

UNIX-Type APIs

The UNIX-type APIs are intended for experienced UNIX programmers who want to do either of the following:

- Create new application programs that run on the AS/400
- Create an AS/400 version of existing application programs that run on other UNIX-based systems

The UNIX-type APIs consist of the following groups.

Environment Variable APIs

Environment variables are character strings of the form `name=value` that are stored in an environment space outside of the program. The strings are stored in a temporary space associated with the job.

Environment variables can be set using the **putenv()** or **Qp0zPutEnv()** function. They can be retrieved using the **getenv()** or **Qp0zGetEnv()** function. The **putenv()** and **getenv()** functions are designed to meet the X/Open** single UNIX specification (formerly Spec 1170). The **Qp0zPutEnv()** and **Qp0zGetEnv()** functions are AS/400 extensions to the industry-standard APIs. They provide the additional capability to store or retrieve a coded character set identifier (CCSID) associated with the string.

After environment variables are set, they exist for the duration of the job. There is no way to remove an environment variable. However, the value can be set to NULL by using a subsequent call to **putenv()** or **Qp0zPutEnv()** and specifying a value of NULL.

The OS/400 support for environment variables does differ from the usual behavior of environment variables on UNIX systems:

- There is no default set of environment variables provided when a job starts. On the AS/400, the `environ` array, which points to the environment variable

strings, is NULL (not initialized) until environment variables are associated with the job.

- On a UNIX system, the exec() function creates a new process and extends the environment variables of the original process to the new process. Although the AS/400 has no exec() function, environment variables *are* extended to a new job created using the Submit Job (SBMJOB) command, provided the first job has environment variables.

To help alleviate these differences, OS/400 provides several nonstandard interfaces that can be used to establish a default set of environment variables.

- The Add Environment Variable (ADDENVVAR) CL command can be used to set an environment variable for a job. Further, a CL program can be written using the ADDENVVAR command that sets several environment variables for a job. Alternatively, the **putenv()** or **Qp0zPutEnv()** function can be used in a similar manner in a C program.
- Other environment variable CL commands, Change Environment Variable (CHGENVVAR) and Work with Environment Variables (WRKENVVAR), or C functions can be used to change or retrieve environment variables for a job.

Integrated File System APIs

The integrated file system is a part of OS/400 that supports stream input/output and storage management similar to personal computer and UNIX operating systems.

The stream file support is designed for efficient use in client/server applications. Stream files are particularly well suited for storing strings of data such as the text of documents, images, audio, and video.

The integrated file system provides a hierarchical directory structure that supports UNIX-based open system standards, such as POSIX** and XPG. This file and directory structure provides the users of PC operating systems with a familiar environment. The integrated file system takes better advantage of the graphical user interface.

In addition to providing a common interface for users and application to access stream files, the integrated file system also provides access to database files, documents and other objects stored on the AS/400.

The integrated file system APIs can perform operations on directories, files, and related objects in the file systems accessed through the integrated file system interface. For more information about the integrated file system, see the *Integrated File System Introduction* book, SC41-5711.

Interprocess Communication APIs

Interprocess communication (IPC) on the AS/400 is made up of three services: message queues, semaphores, and shared memory. The basic purpose of these services is to provide OS/400 processes with a way to communicate with each other through a set of standardized APIs. These C-language functions are based on the definitions in the X/Open single UNIX specification (formerly Spec 1170).

Message queues provide a form of message passing in which any process (given that it has the necessary permissions) can read a message from or write a message to any message queue on the system. There are no requirements that a process be waiting to receive a message from a queue before another process

sends one, or that a message exist on the queue before a process requests to receive one.

A **semaphore** is a synchronization mechanism similar to a mutex or a machine interface (MI) lock. It can be used to control access to shared resources, or used to notify other processes of the availability of resources.

Processes can communicate directly with one another by sharing parts of their memory space and then reading and writing the data stored in the **shared memory**. Synchronization of shared memory is the responsibility of the application program. Semaphores can be used to synchronize shared memory use across processes. Mutexes or condition variables can be used to synchronize shared memory use across threads.

Although each IPC service provides a specific type of interprocess communication, the three services share many similarities. Each service defines a mechanism through which its communications take place. For message queues, that mechanism is a message queue; for semaphores, it is a semaphore set; and for shared memory, it is a shared memory segment. These mechanisms are identified by a unique positive integer, called, respectively, a message queue identifier (*msqid*), a semaphore identifier (*semid*), and a shared memory identifier (*shmid*).

Associated with each identifier is a data structure that contains state information for the IPC mechanism, as well as ownership and permissions information. This structure is similar to a file permissions structure, and is initialized by the process that creates the IPC mechanism. It is then checked by all subsequent IPC operations to determine if the requesting process has the required permissions to perform the operation.

To get an identifier, a process must either create a new IPC mechanism or access an existing mechanism. This is done through the **msgget()**, **semget()**, and **shmget()** functions. Each get operation takes as input a *key* parameter and returns an identifier. Each get operation also takes a *flag* parameter. This *flag* parameter contains the IPC permissions for the mechanism as well as bits that determine whether or not a new mechanism is created.

When a message queue, semaphore set, or shared memory segment is created, the process that creates it determines how it can be accessed. Subsequent IPC operations do a permission test for the calling process before allowing the process to perform the requested operation.

Signal APIs

An X/Open** specification defines a **signal**[1] as “a mechanism by which a process may be notified of, or affected by, an event occurring in the system.” The term signal is also used to refer to the event itself.

A signal is said to be generated when the event that causes the signal first occurs. Examples of such events include the following:

- System-detected errors
- Timer expiration

[1] X/Open CAE Specification System Interface Definitions Issue 4, Number 2, Glossary, page 27. X/Open Company Ltd., United Kingdom, 1994.

- Terminal (work station) activity
- Calling an API such as the X/Open **kill()** function, the American National Standard C **raise()** function, or the ILE **CEESGL** (signal a condition) function.

The **signal action vector** is a list of signal-handling actions for each defined signal. The signal action vector is maintained separately for each process and is inherited from the parent process. The signal action vector specifies the signal-handling actions for both synchronously and asynchronously generated signals.

A signal is said to be **delivered** to a process when the specified signal-handling action for the signal is taken.

The following describes some of the support provided by OS/400 signal management. The set of defined signals is determined by the system. The system specifies the attributes for each defined signal. These attributes consist of a signal number, the initial signal action, and the signal default action. The system also specifies an initial signal blocking mask. The set of defined signals, the signal attributes, and signal blocking mask are referred to as **signal controls**.

A signal can be generated or delivered only to a process that has expressed an interest in signals. An error condition results under the following conditions:

- An attempt is made to generate a signal when the system signal controls have not been initialized.
- An attempt is made to generate a signal for a process that has not been enabled for signals.

A process can express an interest in signals by calling the **Qp0sEnableSignals()** API. In addition, calling particular signal APIs implicitly enables the process for signals.

If the process has not been enabled for signals, the process signal controls are set from signal controls established by the system during IPL (the system signal controls). An error condition results if an attempt is made to enable signals for the process before the system signal controls have been initialized.

Once the process signal controls have been initialized, the user is permitted to change the signal controls for the process.

The attributes for each defined signal are stored in an object called a **signal monitor**. The system supports a maximum of 63 signal monitors for each process. The process signal action vector is a list of signal monitors, one for each defined signal. The signal monitor contains, but is not limited to, the following information:

- Signal action
- Signal default action
- Signal options

The **signal action** defines the action to be taken by the system when a process receives an unblocked signal. The user can change the signal action for a process signal monitor.

The **signal default action** field defines the action to be taken by the system when the signal action is set to *handle using signal default action*. The signal default action for a signal monitor is set in the system signal controls and cannot be changed for a process signal monitor.

The **signal options** specify an additional set of attributes for the signal monitor. The primary use of these options is to specify an additional set of actions to be taken by the system when a signal-catching function is called.

A signal is generated by sending a request to a signal monitor.

The process to receive the signal is identified by a process ID. The **process ID** is used to indicate whether the signal should be sent to an individual process or to a group of processes (known as a process group). The process ID is used to locate an entry in the system-managed process table. A process table entry contains the following information relating to the process:

- Parent process ID
- Process group ID
- Status information

The **parent process** is the logical creator of the process. A **process group** represents a collection of processes that are bound together for some common purpose.

The process sending a signal must have the appropriate authority to the receiving process.

The OS/400 support for signals does differ from the usual behavior of signals on UNIX systems.

For additional information about signal concepts, OS/400 management support, and how signals differ on OS/400 from UNIX systems, see the “Signal Concepts” topic in the book *System API Reference*.

Simple Network Management Protocol (SNMP) APIs

The Simple Network Management Protocol (SNMP) APIs comprise the SNMP subagent APIs and the SNMP manager APIs.

SNMP Subagent APIs: The SNMP subagent APIs can be used to dynamically extend the management information base (MIB) that the system SNMP agent is aware of. The MIB is extended without any change to the SNMP agent itself while the AS/400 is running. Dynamically added MIB subtrees (as supported and defined by a program known as a subagent) provide this capability. You may now extend the remote and automated system management capabilities of the AS/400 within the SNMP framework. So, for example, you could define an SNMP MIB group for your RPG and SQL application.

The **Distributed Protocol Interface (DPI)** is an extension to SNMP agents. DPI permits users to dynamically add, delete, or replace management variables in the local MIB without requiring recompilation of the SNMP agent.

SNMP Manager APIs: SNMP managing applications typically use APIs to establish communication with local or remote SNMP agents, and then call other APIs to retrieve or modify MIB objects managed by those agents. The OS/400 SNMP manager APIs accomplish both of these tasks within the same API. Three manager APIs are provided to perform the SNMP GET, GETNEXT, and SET operations. In general, all three APIs are blocked, that is, once the application calls these APIs, the API constructs a proper SNMP message, delivers it to the proper SNMP agent, waits, decodes the response from the agent, and delivers the information to the application. No processing occurs in the application until the API

delivers this information or times out. The communications mechanism between the manager APIs and agents uses the User Datagram Protocol (UDP). Therefore, both systems need to support UDP.

Sockets APIs

Sockets provides an API for applications that require program-to-program communications. This interface is based on and compatible with Berkeley Software Distributions 4.3. Using sockets, server and client processes can be on the same system or on different systems.

The types of sockets follow:

- Stream sockets, which are connection oriented
- Datagrams, which are connectionless
- Raw sockets, which provide direct access to low-layer protocols
- Sequenced-packet sockets, which are connection oriented

Figure 8-1 shows a typical application flow when the sockets APIs are used.

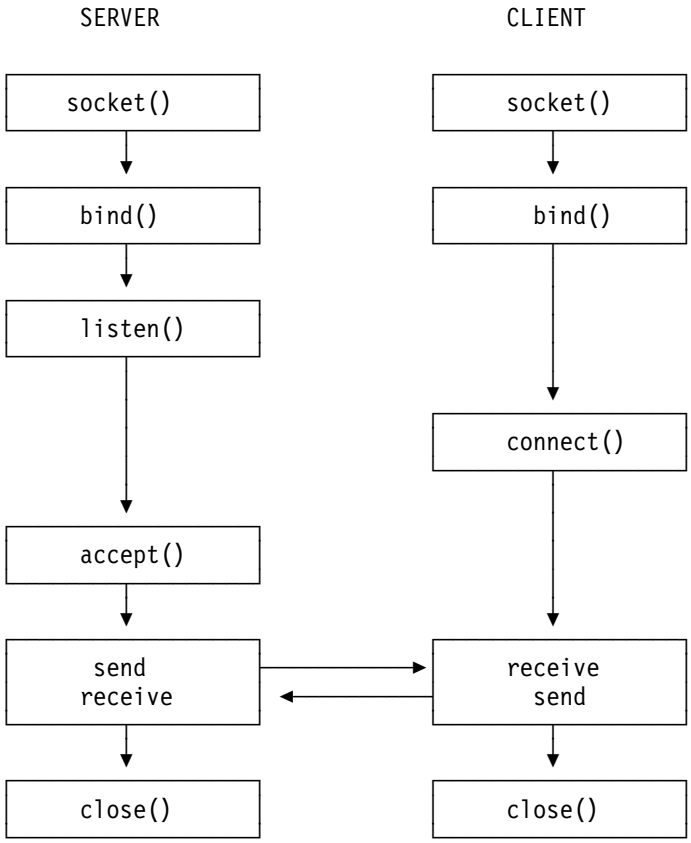


Figure 8-1. Simplified Sequence of Events for a Sockets Program Example

For more information about sockets, see the *Sockets Programming* book, SC41-5422.

Process-Related APIs

The process-related APIs perform process-related or other general operations. Using these APIs, a process can get the process ID of itself, its parent process, or the process group. A process can also check the status for itself, its child processes, or the process group.

For information on APIs to create processes (for example, **spawn()** and **spawnp()**), see the *System API Reference* book.

User Interface APIs

The user interface manager (UIM) APIs are a set of programs that allow the user the ability to use panel groups and create applications that function in the same way. This means using the same standards for function key descriptions, and so forth, as the system panels.

The UIM APIs are used in combination with variables, lists, and panel definitions in a panel group object.

When you design an application program that manipulates the user interface, you can use all UIM panel groups, data description specifications (DDS) display files with UIM help, or DDS display files with help in folders. The following shows the advantages of using UIM versus DDS.

UIM Advantages

The advantages of using UIM over DDS follow:

- Uses the same standards as the AS/400

  There is no need to redefine standards because the applications would work the same way as the system panels. UIM formats the panel for you based on what you want displayed.

- Processes more efficiently from a list panel
  - UIM has the ability to process commands from a list panel. There is no need to call a program to issue the command.
  - UIM has the ability to prompt in the same manner as the AS/400.
  - You can specify a program for UIM to call after the option is selected.
  - Confirmation panels are built into the system.
  - UIM provides more efficient list entry access and update processing.
- Works better with languages that efficiently process structures

  Each type of UIM exit program requires a different set of parameters, thus making it difficult to have one program that processes all the exit program functions.

- Provides for more modular programming techniques

  One program can process all incomplete list exit program calls, one can open all applications, and so forth.

- Has the ability to condition menu options
- Formats and handles scrolling of large areas with no user program intervention

  Examples are data, list, information, menu, and function key areas.

DDS Advantages

The advantages of using DDS over UIM follow:

- Provides more flexibility in screen design

  The user defines in what row and column a field should appear.
- Provides initial formatting with Screen Design Aid (SDA)
- Has the ability to use UIM help or help in folders
- Has the ability to take advantage of graphical operations windows
- Does subfile processing
- Uses edit code (EDTCDE), edit word (EDTWRD), and user-defined editing
- Is faster for smaller applications

  DDS requires less initial setup (that is, the display file may be opened automatically by an HLL program).
- Can also imbed HyperText Markup Language (HTML) tags into the data stream that is sent out

Virtual Terminal APIs

The virtual terminal APIs allow your AS/400 application programs to interact with AS/400 application programs that are performing work station input and output (I/O).

A **virtual terminal** is a device that does not have hardware associated with it. It forms a connection between your application and AS/400 applications, representing a physical work station (possibly on a remote system). The OS/400 licensed program manages the virtual terminal, which directs work station I/O performed by an AS/400 application to the virtual terminal. The virtual terminal APIs allow another AS/400 application, called a **server program**, to work with the data associated with the virtual terminal.

In a distributed systems environment, the requesting program is called a **client**; the answering program is called a **server**. The client and server programs may reside on the same AS/400 system or may be distributed between two different systems. The server program generally runs on behalf of (or in conjunction with) the client program. Together, the server program and the client program allow a work station to be supported as if the work station were connected locally.

Work Management APIs

This group of APIs helps you to better manage the work on your system. The APIs let you keep track of the jobs and the things associated with those jobs. Also, some of the APIs allow you to adjust the performance characteristics.

Work Station Support APIs

The work station support APIs allow you to control the type-ahead characteristics of a work station and to retrieve information about the last output operation to the requester device for the specified interactive job.

Type-ahead, also called **keyboard buffering,** lets the user type data faster than it can be sent to the system. **Attention key buffering** determines how to process the action of pressing an Attention key. If attention key buffering is on, the Atten-

tion key is treated as any other key. If attention key buffering is not on, pressing the Attention key results in sending the information to the system even when other work station input is inhibited.

Miscellaneous APIs

The miscellaneous part of the *System API Reference* includes the following miscellaneous APIs plus the process open list APIs.

Miscellaneous APIs

The miscellaneous APIs include the following:

- Convert Date and Time Format (QWCCVTDT) API

  This API allows you to convert date and time formats from one format to another format.

- Remove All Bookmarks from a Course (QEARMVBM) API

  This API allows you to remove the bookmarks from a Tutorial System Support course.

- Retrieve Data (QPARTVDA) API

  This API retrieves up to 1KB of user data, which was passed to this system with the Start Pass-through (QPASTRPT) API.

- Start Pass-Through (QPASTRPT) API

  This API starts a 5250 pass-through session and optionally passes up to 1KB of user data from the source system to the target system. This data can be accessed on the target system with the Retrieve Data (QPARTVDA) API.

Process Open List APIs

These AS/400 list APIs can improve perceived performance when they create lists. The APIs create and make available to the caller a partial listing of the total set of files, messages, or objects. This list is immediately available to be acted upon, while the remainder of the list is being created. The user does not have to wait for the entire list to be created. Following is a description of the APIs and how they work together.

The process open list APIs are used to access the data returned by the following AS/400 APIs:

- Open List of Job Log Messages (QGYOLJBL)
- Open List of Messages (QGYOLMSG)
- Open List of Objects (QGYOLOBJ)
- Open List of Objects to be Backed Up (QEZOLBKL)
- Open List of Printers (QGYRPRTL)
- Open List of Spooled Files (QGYOLSPL)

The APIs in the previous list are located in their respective sections in the *System API Reference* book; that is, backup and recovery APIs, message handling APIs, object APIs, and print APIs.

Each of these APIs builds a list of the appropriate type and returns the number of records requested by the caller of the API. Also returned from the list building program is a request handle associated with that particular list. This request handle can be used on subsequent calls to the Get List Entry (QGYGTLE) API to

get more records from the list. The request handle is valid until the Close List (QGYCLST) API is used to close the list.

The request handle is also used as input to the following APIs when you need to find a specific entry in the list:

- Find Entry Number in List (QGYFNDE) API, which returns the number of the entry in a list of information for a given key value. This API can be used with lists that have been created by either the QGYOLOBJ or QGYOLSPL API.
- Find Entry Number in Message List (QGYFNDME) API, which returns the number of the entry in the list of message information for a given key value. This API can be used with lists that have been created by either the QGYOLMSG or QGYOLJBL API.
- Find Field Numbers in List (QGYFNDF) API, which returns the number of the entry in a list of information and the value of that entry whenever the value of that field changes.

Chapter 9. Common API Programming Errors

This chapter contains information identified as common programming errors encountered when using APIs within application programs. The chapter design provides two program examples for each common error. The first program example is incorrectly coded and is followed by the correctly coded example. If you encounter errors or problems while working with APIs, these examples may provide ideas or solutions.

Note: Do not assume that an API will do things other than what the *System API Reference* mentions. If the manual does not say specifically that it is allowed, it probably is not.

Figure 9-1 identifies common API programming errors and refers you to examples that show you how to avoid the errors.

Figure 9-1. Common Programming Errors

Task	**Location of Example**
Using the error code parameter	Page 9-2
Defining data structures	Page 9-5
Defining receiver variables	Page 9-10
Defining list entry format lengths	Page 9-14
Using null pointers with OPM APIs	Page 9-18
Defining byte alignment	Page 9-22
Using offsets in user space	Page 9-27
Coding for new function	Page 9-36

Using the Error Code Parameter

The error code parameter provides a way for you to determine if the API encountered any errors.

The examples in this topic present a program used for creating a user space.

Using the Error Code Parameter—Example of Incorrect Coding

The common error shown in the following 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 **2** in the incorrect and correct coding (pages 9-3 and 9-4, respectively).

Both the incorrect and correct coding (**1** on page 9-3 and **1** on page 9-3) 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 (shown at **3** on page 9-3).

```
 ****************************************************************
 *
 *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         1
 *
 * 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
 ** Program does not check the error code parameter  3
 **
C                     SETON                      LR
```

Using the Error Code Parameter—Example of Correct Coding

You can add code to help you discover what errors may be in a program. In the following example program, code has been added to monitor error information passed back in the error code parameter (QUSBN). This code is shown at 4 on page 9-4. The code at 4 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 (shown at 3 on page 9-3).

```
 ****************************************************************
 *
 *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   1
 *
```

```
 * 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                      4
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.

The use of IBM-supplied data structures eliminates having to create your own data structures. For information on IBM-supplied data structures that are contained in library QSYSINC, see “APIs and the QSYSINC Library” on page 2-28.

Defining a Data Structure—Example of Incorrect Coding

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 **5** on page 9-5. The data displayed (JNAME variable) will be incorrect. The correct coding is shown at **6** on page 9-7.

```
*****************************************************************
*
*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    5
```

```
I                                       13  20 USRNAM   5
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
```

Defining A Data Structure—Example of Correct Coding

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 **6** on page 9-7.

```
 *
 *
 ****************************************************************
 *
 *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                    6
 ** 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
```

Defining Receiver Variables

When defining receiver variables, the most common error is to create them too small for the amount of data that they are to receive. Both example programs are coded in RPG and, when run, lists all active jobs on the system.

Defining Receiver Variables—Example of Incorrect Coding

The following example program will fail because the receiver variable has been defined as 50 bytes (shown at **7** on page 9-11), but 60 bytes are being requested to be passed back from the API (shown at **8** in the incorrect and correct programs on pages 9-11 and 9-13, respectively). The correct coding is shown at **9** on page 9-13.

When this happens, other variables are overwritten with unintended data. This causes the other variables to be incorrect. For example, the first 10 characters of QUSBN may be written over with these extra characters. On the call to the next API, the error code parameter may appear to contain meaningless characters that would 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     8
 *
C                     Z-ADD1          X      90
C         X           DOWLEQUSBPS
C                     CALL 'QUSRTVUS'
C                     PARM           SPCNAM
C                     PARM           START
C                     PARM           LENDTA
C                     PARM           RECVR  50   7
C                     PARM           QUSBN
 *
C                     DSPLY          QUSBN
 *
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 Receiver Variables—Example of Correct Coding

The following example program defines a larger receiver variable: 60 bytes. This is shown at position **9** on page 9-13. 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      8
*
C                   Z-ADD1          X      90
C         X         DOWLEQUSBPS
C                   CALL 'QUSRTVUS'
C                   PARM           SPCNAM
C                   PARM           START
C                   PARM           LENDTA
C                   PARM           RECVR  60   9
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

The example programs in this topic show how to code flexibility into your program as it works its way through the formats used by an API.

Defining List Entry Format Lengths—Example of Incorrect Coding

A common error, or trap, when working with list entry format lengths is to hard code the format length into your program. The format length is used by the program to advance to the next list entry in the user space. From release to release, the length of the format may change. Therefore, when the format length changes, your program can be susceptible to being pointed to an incorrect position in the user space and nonsense data placed in the receiver variable.

The program has the length of the list entry format hard coded. This is shown at **10** on page 9-16. 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 **11** on page 9-17.

```
 ****************************************************************
 *
 *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    10
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—Example of Correct Coding

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 **11** on page 9-17. 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       11
 *
 * 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
```

Using Null Pointers with OPM 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 OPM-based APIs are used.

Note: Using NULL with ignored parameters is primarily a consideration with OPM-based APIs. ILE-based APIs 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 may also occur.

Using Null Pointers with OPM APIs—Example of Incorrect Coding

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 **12** on page 9-19. The correct coding is shown at **13** on page 9-21.

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);   12

  /************************************************************/
  /* 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);
}
```

Using Null Pointers with OPM APIs—Example of Correct Coding

The following program specifies that blanks be used as the values for both the member and record format parameters. This coding is shown at **13** on page 9-21 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);     13

  /*********************************************************/
  /* 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 data used with an API is correct. Byte alignment is also essential when APIs are used to retrieve and then print or display data. When byte alignment is off, it causes the API to read the data at some point other than at the beginning of a record.

Defining Byte Alignment—Example of Incorrect Coding

This program illustrates byte alignment while defining a structure. This is shown at **14** on page 9-23. Four-byte alignment is required when using this program.

Variable-length records must begin on a 4-byte boundary. As shown at **14**, 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. **15** on page 9-24 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 **17** on page 9-26.

Note: Not all APIs require a 4-byte boundary. ILE APIs, such as QusAddExitProgram, do.

```
/*************************************************************** */
/*                                                               */
/*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;       14
  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;     15
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 */
```

Defining Byte Alignment—Example of Correct Coding

The following example program shows a CHAR(3) bytes reserved field being added to the structure to maintain 4-byte alignment (shown at **16** on page 9-25). This corresponds to **14** on page 9-23 in the incorrect coding example. The 3 reserved bytes are included in the length of the replace variable-length record. **17** on page 9-26 shows the variable-length record is now 4-byte aligned (record length of 16 is divisible by 4). This corresponds to **15** on page 9-24 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];     16
```

```
  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;     17
  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 */
```

Using Offsets in a User Space

An offset indicates where in a structure that specific information should start. When offsets are correctly used, programs can extract specific pieces of data from a structure and perform actions on that data.

Incorrectly working with offsets can produce errors by API users when coding in a base 1 language such as RPG and COBOL. One way to determine the base of a language is how a programmer specifies the first element of an array. 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 while the APIs produce information using a base of 0. To compensate for APIs producing information at base 0, the API user must add 1 to all decimal and hexadecimal offsets to formats that are contained in the *System API Reference* book.

Using Offsets in a User Space—Example of Incorrect Coding

The point for beginning to read a user space is shown at **18** on page 9-30. The data is read and placed into a user space. However, the data in the user space is incorrect because the starting 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 **19** on page 9-34.

```
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  (C) Copyright IBM Corp. 1994,1994
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    18
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
```

Using Offsets in a User Space—Example of Correct Coding

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 **19** on page 9-34.

```
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  (C) Copyright IBM Corp. 1994,1994
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      19
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 Function

New function from IBM can cause programs to fail if the programs do not allow for the handling of a new function.

The example programs in the following topics create a list of all objects that adopt authority and then process the objects based on their object type. The new function added is the addition of another object type, *SRVPGM, that can adopt owner authority.

A general theme of this example is never to assume that the values returned by an API are static. OS/400 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 should discretely check for these values because *MAYBE might be valid in the future. Similarly, if your application assumes a particular integer output has a positive nonzero value (an offset for instance), you should check for a positive nonzero value because future releases could return a negative value to indicate new function.

Coding for New Function—Example of Incorrect Coding

In this example program, a check is made to determine the object type. This is shown at **20** on page 9-40. 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 **23** on page 9-47.

```
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 AS/400.
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 AS/400.
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   (shown at  21
D* in the incorrect and correct programs on pages
D* 9-37 and 9-45, respectively)
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  (C) Copyright IBM Corp.  1994, 1994
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)     21
```

```
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  (shown at 22
D* in the incorrect and correct coding on
D* pages 9-39 and 9-46, respectively)
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)     22
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)    22
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                                          20
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                                         ↓
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
```

Coding for New Function—Example of Correct Coding

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 **23** on page 9-47.

```
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 AS/400.
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   (shown at  21
D* in the incorrect and correct programs on pages
D* 9-37 and 9-45, respectively)
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  (C) Copyright IBM Corp.  1994, 1994
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)        21
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  (shown at 22
D* in the incorrect and correct coding on
D* pages 9-39 and 9-46,
D* respectively)
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)      22
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)      22
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                                        23
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*                                                      ↓
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
```

Appendix A. Performing Tasks Using APIs—Examples

This appendix contains the following examples of using multiple APIs to perform tasks:

- Packaging your own software products
- Retrieving a file description to a user space
- Using data queues versus user queues

Packaging Your Own Software Products

You can define, create, distribute, and maintain your own product using APIs. The following demonstrates how you can use the APIs to package a product similar to the way IBM packages products.

The example product being packaged in this example 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:

Figure A-1. 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

Note:

1. Machine readable material
2. Machine readable information
3. Command processing program

To package a product, first you create all of the objects (numbers 1 through 11 and number 15 in Figure A-1) that will comprise your product. (“CL Program for Creating Objects and Library for Packaging a Product” on page A-2 shows the code that creates the objects.) After your objects are created, you do the steps listed in “Program for Packaging a Product—OPM RPG Example” on page A-3.

The following figure is an overview of the steps required to create a product. An explanation is given of the numbers in Figure A-2. The same numbers also appear in the code.

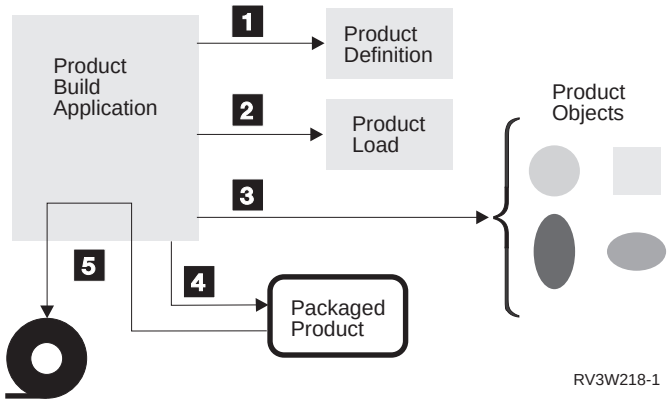


Figure A-2. Steps for Creating a Software Product

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.

CL Program for Creating Objects and Library for Packaging a Product

The following CL program creates objects 1 through 11 and 15 in Figure A-1 on page A-1.

```
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
```

Program for Packaging a Product—OPM RPG Example

The following program creates objects 12 through 14 in Figure A-1 on page A-1.

```
     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  (C) Copyright IBM Corp. 1994,1994
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.

Once the product is built, you can do the following:

- 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).

Note: For examples of the software product example program in additional languages, see “Program for Packaging a Product—Examples” on page B-129.

Retrieving a File Description to a User Space—ILE C Example

The following programming example shows an application that uses a user space as a receiver variable by retrieving a file description to a user space. This approach is possible only if you use an HLL that is able to work with pointers. The application accepts the following parameters:

- User space name and library
- File name and library
- Record format

The following shows the sequence of steps to retrieve a file description to a user space:

1. The application 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 application calls the Retrieve File Description API to retrieve the file definition template and uses the user space as the receiver variable.

This example uses 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.

```
/*****************************************************************/
/* 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 example 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” in the *System API Reference* book.

Using Data Queues versus User Queues

Data queues and user queues both provide a means for one or more processes to communicate asynchronously. The queues can be processed FIFO (first-in first-out), LIFO (last-in first-out), or by key. If user queues and data queues supply the same function, which one should you choose for your implementation? The following is a comparison of the two and an insight into when you should use one queue rather than the other.

First, your programming experience is an important consideration in selecting a queue type. If you are skilled in C or MI programming, you may want to select the user queue. C and MI are the only languages that can use MI instructions, which, as discussed later, has a bearing on performance. If your expertise is in COBOL or RPG, then you should choose the data queue. You cannot implement a user queue in COBOL or RPG because neither of these languages can use MI instructions.

Next, performance plays an important part in determining what type of queue to use. As stated in Chapter 1, “Application Programming Interface—Overview” on page 1-1, 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. (See “Domain Concepts” on page 2-26 for more information on QALWUSRDMN.) 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 is a summary to help you select the type of queue that is right for your program:

- Use user queues when:
  - You have a programming background in or prefer to program in C or 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 COBOL 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.

Data Queue—ILE C Example

The following program illustrates how to use APIs to create and manipulate a data queue.

```
/*********************************************************************/
/*                                                                   */
/*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)");

}
```

Note: For examples of the data queue program in additional languages, see “Data Queue—Examples” on page B-165.

User Queue—ILE C Example

The following program illustrates how to use APIs to create and manipulate a user queue.

```
/*****************************************************************/
/*                                                               */
/*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)");
}
```

Appendix B. Original Examples in Additional Languages

This appendix contains examples from the following areas of this book. The program examples in this appendix are rewritten from the original examples into other programming languages.

- Chapter 3, “Common Information across APIs—Basic (OPM) Example”
- Chapter 4, “Common Information across APIs—Advanced (ILE) Example”
- Chapter 5, “List APIs”
- Chapter 6, “Original Program Model (OPM) and Integrated Language Environment (ILE) Differences”
- Appendix A, “Performing Tasks Using APIs—Examples”

This appendix also contains examples written in ILE C, ILE COBOL, and ILE RPG for using the integrated file system.

The following tables describe the example tasks and refer you to the corresponding programming language topic. In these tables, **Yes** means the task can be accomplished in the language identified but an example is not provided. **No** means the task cannot be accomplished in the language identified.

Figure B-1. Original Program Model (OPM) API Examples from Chapter 3

	Programming Language and Location of Example				
Task	**ILE C**	**OPM COBOL**	**ILE COBOL**	**OPM RPG**	**ILE RPG**
Retrieving the HOLD Parameter (Exception Messages)	B-2	B-4	B-4	3-6	B-6
Handling Error Conditions	No	No	No	3-8	B-8
Retrieving the HOLD Parameter (Error Code Structure)	B-10	B-12	B-12	3-11	B-14
Accessing the HOLD Attribute	B-16	B-18	B-18	3-17	B-21
Accessing a Field Value (Initial Library List)	B-22	B-25	B-25	3-19	B-29
Using Keys with List Spooled Files API	B-33	B-38	B-38	3-24	B-42

Figure B-2. Integrated Language Environment (ILE) API Examples from Chapter 4

	Programming Language and Location of Example				
Task	**ILE C**	**OPM COBOL**	**ILE COBOL**	**OPM RPG**	**ILE RPG**
Register Exit Point and Add Exit Program	4-9	B-47	B-50	B-54	B-58
Retrieve Exit Point and Exit Program Information	4-13	B-61	B-66	B-71	B-75
Remove Exit Program and Deregister Exit Point	4-19	B-85	B-87	B-90	B-92

Figure B-3. List API Examples from Chapter 5

	Programming Language and Location of Example				
Task	**ILE C**	**OPM COBOL**	**ILE COBOL**	**OPM RPG**	**ILE RPG**
Listing Objects	B-94	B-101	B-101	5-4	B-106

Figure B-4 (Page 1 of 2). Pointer API Examples from Chapter 6

	Programming Language and Location of Example				
Task	**ILE C**	**OPM COBOL**	**ILE COBOL**	**OPM RPG**	**ILE RPG**
Logging Software Error (OPM API without Pointers)	6-2	B-112	N/A	B-116	B-119

Figure B-4 (Page 2 of 2). Pointer API Examples from Chapter 6					
	Programming Language and Location of Example				
Task	**ILE C**	**OPM COBOL**	**ILE COBOL**	**OPM RPG**	**ILE RPG**
Reporting Software Error (ILE API with Pointers)	6-7	N/A	B-122	N/A	B-126

Figure B-5. Performing Tasks Using API Examples from Appendix A					
	Programming Language and Location of Example				
Task	**ILE C**	**OPM COBOL**	**ILE COBOL**	**OPM RPG**	**ILE RPG**
Program for Packaging a Product	B-129	B-136	B-136	A-3	B-144
Retrieving a File Description to a User Space	A-11	B-152	B-152	No	B-155
Working with Data Queues	A-16	B-165	B-165	B-169	B-172
Working with User Queues	A-17	No	No	No	No

Figure B-6. UNIX-Type API Examples					
	Programming Language and Location of Example				
Task	**ILE C**	**OPM COBOL**	**ILE COBOL**	**OPM RPG**	**ILE RPG**
Using the Integrated File System	B-175	No	B-178	No	B-183

Original Program Model (OPM) APIs—Examples

This topic includes the examples in Chapter 3, “Common Information across APIs—Basic (OPM) Example.”

Retrieving the Hold Parameter (Exception Message)—ILE C Example

Refer to “Retrieving the Hold Parameter (Exception Message)—OPM RPG Example” on page 3-6 for the original example.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*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 */
```

Retrieving the Hold Parameter (Exception Message)—ILE COBOL Example

Refer to “Retrieving the Hold Parameter (Exception Message)—OPM RPG Example” on page 3-6 for the original example. The following program also works for OPM COBOL.

```
       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.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
*
 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.
```

Retrieving the Hold Parameter (Exception Message)—ILE RPG Example

Refer to “Retrieving the Hold Parameter (Exception Message)—OPM RPG Example” on page 3-6 for the original example.

```
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*
```

Handling Error Conditions—ILE RPG Example

Refer to “Handling Error Conditions—OPM RPG Example” on page 3-8 for the original example. This example 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
```

Retrieving the Hold Parameter (Error Code Structure)—ILE C Example

Refer to “Retrieving the Hold Parameter (Error Code Structure)—OPM RPG Example” on page 3-11 for the original example.

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/*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>  14   /* 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; 15

  /********************************************************************/
  /* 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) 13
  {
    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 */
```

Retrieving the Hold Parameter (Error Code Structure)—ILE COBOL Example

Refer to “Retrieving the Hold Parameter (Error Code Structure)—OPM RPG Example” on page 3-11 for the original example. The following program also works for OPM COBOL.

```
 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.
*
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
*
 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.  14
*
* 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.  15
*
* 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  13
*
* 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.
```

Retrieving the Hold Parameter (Error Code Structure)—ILE RPG Example

Refer to “Retrieving the Hold Parameter (Error Code Structure)—OPM RPG Example” on page 3-11 for the original example.

```
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   14
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)  15
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       13
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
```

Accessing the HOLD Attribute—ILE C Example

Refer to “Accessing the HOLD Attribute—OPM RPG Example” on page 3-17 for the original example.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*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 */
```

Accessing the HOLD Attribute—ILE COBOL Example

Refer to “Accessing the HOLD Attribute—OPM RPG Example” on page 3-17 for the original example. The following example also works for OPM COBOL.

```
       IDENTIFICATION DIVISION.
      ******************************************************************
      ******************************************************************
      *
      *Program Name:          JOBDAPI
      *
      *Programming Language:  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.
      *
```

```
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
*
 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.
```

Accessing the HOLD Attribute—ILE RPG Example

Refer to “Accessing the HOLD Attribute—OPM RPG Example” on page 3-17 for the original example.

```
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
```

Accessing a Field Value (Initial Library List)—ILE C Example

Refer to “Accessing a Field Value (Initial Library List)—OPM RPG Example” on page 3-19 for the original example.

```
/*********************************************************************/
/*********************************************************************/
/*                                                                   */
/*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]; 7
        } 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[610]; 19  20
        } 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 = 1000;

  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 */
```

Accessing a Field Value (Initial Library List)—ILE COBOL Example

Refer to “Accessing a Field Value (Initial Library List)—OPM RPG Example” on page 3-19 for the original example. The following program also works for OPM COBOL.

```
       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.
      *
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
         SOURCE-COMPUTER. IBM-AS400.
         OBJECT-COMPUTER. IBM-AS400.
      *
       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  (C) Copyright IBM Corp. 1994,1994
*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(01000).
```

```
 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. 19
     05  NUMBER-LIBS-IN-LIB-LIST             PIC S9(00009) BINARY. 20
     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 1000.
*
     MOVE 1000 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.
```

Accessing a Field Value (Initial Library List)—ILE RPG Example

Refer to “Accessing a Field Value (Initial Library List)—OPM RPG Example” on page 3-19 for the original example.

```
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  (C) Copyright IBM Corp. 1994,1994
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          1000
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 19
D*                                          Offset Initial Lib List
D QWDNLILL              365    368B 0 20
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
```

Using Keys with List Spooled Files API—ILE C Example

Refer to “Using Keys with List Spooled Files API—Example” on page 3-24 for the original example.

```
/*****************************************************************/
/*                                                               */
/*   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
```

Using Keys with List Spooled Files API—ILE COBOL Example

Refer to “Using Keys with List Spooled Files API—Example” on page 3-24 for the original example. The following program also works for OPM COBOL.

```
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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).
```

Using Keys with List Spooled Files API—ILE RPG Example

Refer to “Using Keys with List Spooled Files API—Example” on page 3-24 for the original example.

```
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'
```

Integrated Language Environment (ILE) APIs—Examples

This section includes the examples in Chapter 4, “Common Information across APIs—Advanced (ILE) Example.”

Register Exit Point and Add Exit Program—OPM COBOL Example

Refer to “Register Exit Point and Add Exit Program—ILE C Example” on page 4-9 for the original example.

```
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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.
```

Register Exit Point and Add Exit Program—ILE COBOL Example

Refer to “Register Exit Point and Add Exit Program—ILE C Example” on page 4-9 for the original example.

```
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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.
```

Register Exit Point and Add Exit Program—OPM RPG Example

Refer to “Register Exit Point and Add Exit Program—ILE C Example” on page 4-9 for the original example.

```
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  68  710EPGMSZ
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
```

Register Exit Point and Add Exit Program—ILE RPG Example

Refer to “Register Exit Point and Add Exit Program—ILE C Example” on page 4-9 for the original example.

```
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:V_OFFSET+12) = BINARY_C
C                   EXSR      CALC_VOFF
C                   EVAL      QUSCK = 8
C                   EVAL      QUSLD = 50
C                   EVAL      %SUBST(RECS: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:V_OFFSET+12) = '1'
C                   EXSR      CALC_VOFF
C                   EVAL      QUSCK = 3
C                   EVAL      QUSLD = 4
C                   EVAL      BINARY = 37
C                   EVAL      %SUBST(RECS: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: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
```

Retrieve Exit Point and Exit Program Information—OPM COBOL Example

Refer to “Retrieve Exit Point and Exit Program Information—ILE C Example” on page 4-13 for the original example.

```
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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:).
```

Retrieve Exit Point and Exit Program Information—ILE COBOL Example

Refer to “Retrieve Exit Point and Exit Program Information—ILE C Example” on page 4-13 for the original example.

```
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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:).
```

Retrieve Exit Point and Exit Program Information—OPM RPG Example

Refer to “Retrieve Exit Point and Exit Program Information—ILE C Example” on page 4-13 for the original example.

```
     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: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: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: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
```

Retrieve Exit Point and Exit Program Information—ILE RPG Example

Refer to "Retrieve Exit Point and Exit Program Information—ILE C Example" on page 4-13 for the original example.

```
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  (C) Copyright IBM Corp. 1994,1994
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
```

Remove Exit Program and Deregister Exit Point—OPM COBOL Example

Refer to “Remove Exit Program and Deregister Exit Point—ILE C Example” on page 4-19 for the original example.

```
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
```

```
  OBJECT-COMPUTER. IBM-AS400.
 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
*
```

Remove Exit Program and Deregister Exit Point—ILE COBOL Example

Refer to “Remove Exit Program and Deregister Exit Point—ILE C Example” on page 4-19 for the original example.

```
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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
*
```

Remove Exit Program and Deregister Exit Point—OPM RPG Example

Refer to “Remove Exit Program and Deregister Exit Point—ILE C Example” on page 4-19 for the original example.

```
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
```

Remove Exit Program and Deregister Exit Point—ILE RPG Example

Refer to “Remove Exit Program and Deregister Exit Point—ILE C Example” on page 4-19 for the original example.

```
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
```

List Object API—Examples

This section includes the examples in “List Object API—OPM RPG Example” on page 5-4.

List Object API—ILE C Example

Refer to “List Object API—OPM RPG Example” on page 5-4 for the original example. This example uses includes from the QSYSINC library.

```
/*****************************************************************/
/*****************************************************************/
/*                                                               */
/* 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 */
```

List Object API—ILE COBOL Example

Refer to “List Object API—OPM RPG Example” on page 5-4 for the original example. The following program also works for OPM COBOL.

```
       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.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
         SOURCE-COMPUTER. IBM-AS400.
         OBJECT-COMPUTER. IBM-AS400.
       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.
         09  TEXT1         PIC  X(08) VALUE "Object: ".
         09  NAME          PIC  X(10).
         09  TEXT2         PIC  X(10) VALUE " Library: ".
         09  LIBRARY       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.
```

List Object API—ILE RPG Example

Refer to “List Object API—OPM RPG Example” on page 5-4 for the original example.

```
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'
```

OPM API without Pointers—Examples

This section includes the examples in “Reporting Software Error (ILE API with Pointers)—ILE C Example” on page 6-7.

Logging Software Error (OPM API without Pointers)—OPM COBOL Example

Refer to “Logging Software Error (OPM API without Pointers)—ILE C Example” on page 6-2 for the original example. This example uses two programs: CBLERR1 causes the error, and ERRHDL1 shows how to log the software error using the QPDLOGER API.

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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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 error".
*
* End the current run unit
*
     MOVE "C" TO SYS-OPTION.
     STOP RUN.
*
* End of MAINLINE
*
```

Logging Software Error (OPM API without Pointers)—OPM RPG Example

Refer to “Logging Software Error (OPM API without Pointers)—ILE C Example” on page 6-2 for the original example.

```
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
```

Logging Software Error (OPM API without Pointers)—ILE RPG Example

Refer to “Logging Software Error (OPM API without Pointers)—ILE C Example” on page 6-2 for the original example.

```
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
```

ILE API with Pointers—Examples

This section includes the examples in “Reporting Software Error (ILE API with Pointers)—ILE C Example” on page 6-7.

Reporting Software Error (ILE API with Pointers)—ILE COBOL Example

Refer to “Reporting Software Error (ILE API with Pointers)—ILE C Example” on page 6-7 for the original example. This example uses two programs: CBLERR2 causes the error, and ERRHDL2 shows how to report the software error using the QPDLOGER API.

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.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
         SOURCE-COMPUTER. IBM-AS400.
         OBJECT-COMPUTER. IBM-AS400.
         SPECIAL-NAMES.
         LINKAGE TYPE PROCEDURE FOR "QlnSetCobolErrorHandler".
       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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
   SPECIAL-NAMES.
   LINKAGE TYPE PROCEDURE FOR "QpdReportSoftwareError".
 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
*
```

Reporting Software Error (ILE API with Pointers)—ILE RPG Example

Refer to “Reporting Software Error (ILE API with Pointers)—ILE C Example” on page 6-7 for the original example.

```
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
```

Program for Packaging a Product—Examples

This section includes the examples in “Packaging Your Own Software Products” on page A-1.

Program for Packaging a Product—ILE C Example

Refer to “Program for Packaging a Product—OPM RPG Example” on page A-3 for the original example.

```
/*****************************************************************/
/* 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();

}
```

Program for Packaging a Product—ILE COBOL Example

Refer to “Program for Packaging a Product—OPM RPG Example” on page A-3 for the original example. 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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.
```

Program for Packaging a Product—ILE RPG Example

Refer to “Program for Packaging a Product—OPM RPG Example” on page A-3 for the original example.

```
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
```

Retrieving a File Description to a User Space—Examples

This section includes the examples in “Retrieving a File Description to a User Space—ILE C Example” on page A-11.

Retrieving a File Description to a User Space—ILE COBOL Example

Refer to “Retrieving a File Description to a User Space—ILE C Example” on page A-11 for the original example. The following program also works with OPM COBOL.

```
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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.
```

Retrieving a File Description to a User Space—ILE RPG Example

Refer to “Retrieving a File Description to a User Space—ILE C Example” on page A-11 for the original example.

```
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
```

Data Queue—Examples

This section includes the examples in “Using Data Queues versus User Queues” on page A-15.

Data Queue—ILE COBOL Example

Refer to “Data Queue—ILE C Example” on page A-16 for the original example. The following program also works with OPM COBOL.

```
 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.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
 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.
```

Data Queue—OPM RPG Example

Refer to “Data Queue—ILE C Example” on page A-16 for the original example.

```
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
```

Data Queue—ILE RPG Example

Refer to "Data Queue—ILE C Example" on page A-16 for the original example.

```
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
```

UNIX-Type APIs—Examples

The simple example program on the following pages illustrates the use of several integrated file system functions. The 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.

Using the Integrated File System—ILE C Example

This example program uses the integrated file system from ILE C.

```
/*********************************************************************/
/*                                                                   */
/* 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;
}
```

Using the Integrated File System—ILE COBOL Example

This example program uses the integrated file system from ILE COBOL.

```
       PROCESS NOMONOPRC.
       IDENTIFICATION DIVISION.
      *************************************************************
      *************************************************************
      *
```

```
*  Language:     COBOL
*
*  Description:  Demonstrate use of integrated file system
*                from ILE COBOL
*
*****************************************************************
*
*****************************************************************
 PROGRAM-ID. IFS.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SOURCE-COMPUTER. IBM-AS400.
   OBJECT-COMPUTER. IBM-AS400.
   SPECIAL-NAMES.
*  LINKAGE TYPE PROCEDURE FOR "geterrno",
   LINKAGE TYPE PROCEDURE FOR "getuid",
   LINKAGE TYPE PROCEDURE FOR "getcwd",
   LINKAGE TYPE PROCEDURE FOR "open",
   LINKAGE TYPE PROCEDURE FOR "write",
   LINKAGE TYPE PROCEDURE FOR "close",
   LINKAGE TYPE PROCEDURE FOR "read",
   LINKAGE TYPE PROCEDURE FOR "unlink".
 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.
```

Using the Integrated File System—ILE RPG Example

This example program uses the integrated file system from ILE RPG.

```
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:1: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
```

Bibliography

This bibliography lists printed information that you need to use the OS/400 APIs, background information for the functions the APIs perform, and other information relevant to specific types of applications. The books are grouped in these categories:

- General-purpose books
- OS/400 API books
- Programming language books

If you want more information on a topic while you are using this guide, see the *Publications Reference*, SC41-5003, for related AS/400 publications.

General-Purpose Books

These books provide general-purpose and background information for the OS/400 licensed program:

- *CL Programming*, SC41-5721, discusses OS/400 functions and concepts that are relevant to programming.
- *CL Reference*, SC41-5722, provides a description of the AS/400 control language (CL) and its commands. Each command description includes a syntax diagram, parameters, default values, keywords, and an example.
- *AS/400 Licensed Internal Code Diagnostic Aids – Volume 1*, LY44-5900, and *AS/400 Licensed Internal Code Diagnostic Aids – Volume 2*, LY44-5901, provide a list of available object types in hexadecimal format for use with the object APIs.
- *ILE Concepts*, SC41-5606, describes the concepts and terminology of the Integrated Language Environment of the OS/400 operating system.
- *Integrated File System Introduction*, SC41-5711, describes the concepts of the integrated file system and briefly describes the user interfaces and programming support for interacting with the integrated file system.
- *Printer Device Programming*, SC41-5713, provides information to help you understand and control printing. This book provides specific information on printing elements and concepts of the AS/400 system, printer file and print spooling support for printing operations, and printer connectivity.
- *Guide to Enabling C2 Security*, SC41-0103, provides information about planning, installing, setting up, and managing your AS/400 system to meet the requirements for C2 security. C2 is a level of security defined by the United States Department of Defense.
- *Security – Reference*, SC41-5302, provides technical information about OS/400 security.
- *System Manager Use*, SC41-5321, provides information about the commands and functions available when the System Manager for AS/400 licensed program is installed on one or more AS/400 systems in a network. *System Manager Use* describes packaging software products so that they can be distributed, installed, and serviced the same way IBM licensed programs are managed.

OS/400 API Books

These books contain OS/400 APIs:

- *Common Programming APIs Toolkit/400 Reference*, SC41-4802, describes considerations for creating, running, and debugging Common Programming APIs Toolkit/400 (CPA) programs and provides details on APIs supported by CPA. This book also includes examples of CPA programs. CPA is an optionally installable feature of OS/400.
- *CPI Communications Reference*, SC26-4399, provides information about writing applications that adhere to the Systems Application Architecture (SAA) Communications interface. The manual defines the elements of the SAA Communications Common Programming Interface (CPI), which provides a programming interface that allows program-to-program communications using IBM's Systems Network Architecture (SNA) logical unit 6.2 (LU6.2).
- *DB2 for AS/400 Query Management Programming*, SC41-5703, provides information on how to determine database files to be queried for a report, define a structured query language (SQL) query definition, define a report form definition, and use and write procedures that use query management commands. It also includes information on how to use the query management global variable support and understand the relationship between the OS/400 query management and the Query/400 licensed program.
- *GDDM Programming Guide*, SC41-0536, provides information about using OS/400 graphical data display manager (GDDM) to write graphics application programs.
- *Machine Interface Functional Reference*, SC41-5810, is a comprehensive reference to machine interface (MI) instructions.
- *PrintManager API Reference*, S544-3699, provides information the user needs to create and maintain

print descriptors used by the PrintManager interface.

- *REXX/400 Programmer's Guide*, SC41-5728, explains REXX/400 programming concepts and discusses considerations in using this language on the AS/400 system. It also describes REXX APIs and provides examples that you can use to learn REXX/400.
- *REXX/400 Reference*, SC41-5729, provides an overview of the REXX/400 concepts and includes information about keyword instruction syntax, function syntax, numerics, arithmetic, conditions, input and output streams, testing, and double-byte character set (DBCS) support. The book also describes REXX APIs.
- *Sockets Programming*, SC41-5422, describes the sockets programming functions available on AS/400 systems and provides reference information on the sockets programming interface.
- *System API Reference*, SC41-5801, describes OS/400 APIs. It is intended for experienced application programmers who are developing system-level and other OS/400 applications.
- *Ultimedia System Facilities Programming*, SC41-4652, provides information to programmers for using Ultimedia System Facilities APIs to add multimedia interfaces to existing applications and to develop AS/400 programmable workstation (PWS) multimedia applications.

Programming Language Books

You might refer to these programming language books while writing applications with the OS/400 APIs:

- *COBOL/400 User's Guide*, SC09-1812, provides information needed to design, write, test, and maintain COBOL programs on the AS/400 system.
- *ILE C/400 Programmer's Guide*, SC09-2069, provides information on how to develop applications using the ILE C language. It includes information about creating, running, and debugging programs. It also includes programming considerations for interlanguage program and procedure calls, locales, exception handling, database files, externally described files, and device files. Some performance tips are also described. An appendix includes information on migrating source code from extended program model (EPM) C/400 or System C/400 to ILE C.
- *ILE C/400 Programmer's Reference*, SC09-2070, provides information about how to write programs that adhere to the Systems Application Architecture C Level 2 definition and use ILE C specific functions such as record I/O. It also provides information on ILE C machine interface library functions.
- *ILE COBOL/400 Programmer's Guide*, SC09-2072, describes how to write, compile, bind, run, debug, and maintain ILE COBOL programs on the AS/400 system. It provides programming information on how to call other ILE COBOL and non-ILE COBOL programs, share data with other programs, use pointers, and handle exceptions. It also describes how to perform input/output operations on externally attached devices, database files, display files, and ICF files.
- *ILE COBOL/400 Reference*, SC09-2073, describes the ILE COBOL programming language. It provides information on the structure of the ILE COBOL programming language and on the structure of an ILE COBOL source program. It also describes all Identification Division paragraphs, Environment Division clauses, Data Division paragraphs, Procedure Division statements, and Compiler-Directing statements.
- *ILE RPG/400 Programmer's Guide*, SC09-2074, is a guide for using the ILE RPG programming language, which is an implementation of the RPG IV language in the Integrated Language Environment (ILE) on the AS/400 system. It includes information on creating and running programs, with considerations for procedure calls and interlanguage programming. The guide also covers debugging and exception handling and explains how to use AS/400 files and devices in RPG programs. Appendixes include information on migration to RPG IV and sample compiler listings. It is intended for people with a basic understanding of data processing concepts and of the RPG language.
- *ILE RPG/400 Reference*, SC09-2077, provides information needed to write programs for the AS/400 system using the ILE RPG programming language. This book describes, position by position and keyword by keyword, the valid entries for all RPG specifications, and provides a detailed description of all the operation codes and built-in functions. This book also contains information on the RPG logic cycle, arrays and tables, editing functions, and indicators.
- *RPG/400 Reference*, SC09-1817, provides information needed to write programs for the AS/400 system using the RPG programming language. This book describes, position by position, the valid entries for all RPG specifications, and provides a detailed description of all the operation codes. This book also contains information on the RPG logic cycle, arrays and tables, editing functions, and indicators.
- *RPG/400 User's Guide*, SC09-1816, provides information needed to write, test, and maintain RPG programs on the AS/400 system. The book provides information on data organizations, data formats, file processing, multiple file processing, automatic report function, RPG command statements, testing

and debugging functions, application design techniques, problem analysis, and compiler service information. The differences between the System/38 RPG III, System/38 compatible RPG, and RPG are identified.

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Reader Comments—We'd Like to Hear from You!

AS/400 Advanced Series
System API Programming
Version 4

Publication No. SC41-5800-00

Overall, how would you rate this manual?

	Very Satisfied	Satisfied	Dissatis-fied	Very Dissatis-fied
Overall satisfaction				

How satisfied are you that the information in this manual is:

Accurate				
Complete				
Easy to find				
Easy to understand				
Well organized				
Applicable to your tasks				
THANK YOU!				

Please tell us how we can improve this manual:

May we contact you to discuss your responses? __ Yes __ No
Phone: (____) _____ Fax: (____) _____ Internet: _____

To return this form:

- Mail it
- Fax it
  United States and Canada: **800+937-3430**
  Other countries: **(+1)+507+253-5192**
- Hand it to your IBM representative.

Note that IBM may use or distribute the responses to this form without obligation.

Name

Address

Company or Organization

Phone No.

Reader Comments—We'd Like to Hear from You!
SC41-5800-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DEPT 542 IDCLERK
IBM CORPORATION
3605 HWY 52 N
ROCHESTER MN 55901-9986

Fold and Tape **Please do not staple** Fold and Tape

SC41-5800-00

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC41-5800-00

Spine information:

AS/400 Advanced Series System API Programming *Version 4*