



IBM Systems - iSeries

Integrated operating environments  
i5/OS PASE

*Version 5 Release 4*







IBM Systems - iSeries

Integrated operating environments  
i5/OS PASE

*Version 5 Release 4*

**Note**

Before using this information and the product it supports, be sure to read the information in "Notices," on page 57.

**Sixth Edition (February 2006)**

This edition applies to version 5, release 4, modification 0 of IBM i5/OS (product number 5722-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 2000, 2006. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

|  |          |
|--|----------|
| <b>i5/OS PASE</b> . . . . .  | <b>1</b> |
| What's new for V5R4 . . . . .  | 1        |
| Printable PDF . . . . .  | 2        |
| Get started with i5/OS PASE . . . . .  | 2        |
| What is i5/OS PASE? . . . . .  | 2        |
| When is i5/OS PASE a useful option for<br>application development? . . . . . | 4        |
| Install i5/OS PASE . . . . .   | 5        |
| Plan for i5/OS PASE. . . . .   | 5        |
| Prepare programs to run in i5/OS PASE . . . . .                              | 7        |
| Analyze your program's compatibility with i5/OS<br>PASE . . . . .            | 7        |
| Compile your AIX source . . . . .  | 8        |
| Copy the i5/OS PASE program to your iSeries<br>server . . . . .              | 12       |
| Customize i5/OS PASE programs to use i5/OS<br>functions . . . . .            | 15       |

|  |    |
|--|----|
| Use i5/OS PASE programs in the i5/OS<br>environment . . . . .                | 18 |
| Run i5/OS PASE programs and procedures. . . . .                              | 18 |
| Call i5/OS programs and procedures from your<br>i5/OS PASE programs. . . . . | 28 |
| How i5/OS PASE programs interact with i5/OS . . . . .                        | 39 |
| Debug your i5/OS PASE programs . . . . .                                     | 53 |
| Optimize performance. . . . .  | 54 |
| Examples . . . . .   | 54 |
| Related information for i5/OS PASE . . . . .                                 | 55 |
| Code license and disclaimer information. . . . .                             | 56 |

|   |           |
|---|-----------|
| <b>Appendix. Notices</b> . . . . .          | <b>57</b> |
| Programming Interface Information . . . . . | 58        |
| Trademarks . . . . .                        | 59        |
| Terms and conditions . . . . .              | 59        |



---

## i5/OS PASE

IBM® i5/OS™ Portable Application Solutions Environment (i5/OS PASE) allows you to port IBM AIX® applications to the IBM iSeries™ server with minimal effort.

i5/OS PASE provides an integrated runtime environment that allows you to run selected applications without the complexity of managing operating systems, such as AIX or Linux®. i5/OS PASE also provides industry-standard and de facto-standard shells and utilities that provide you with a powerful scripting environment.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 56.

---



## What’s new for V5R4

This page highlights the changes to the i5/OS PASE in this release.

- | • i5/OS PASE for V5R4M0 is derived from AIX 5.3 (versus AIX 5.2 for i5/OS PASE V5R3M0)
- | • The following compiler products announced support to run on V5R4M0 of i5/OS PASE:
  - | – IBM XL C/C++ Enterprise Edition for AIX, V7.0
  - | – IBM XL C Enterprise Edition for AIX, V7.0
  - | – IBM XL Fortran Enterprise Edition for AIX, V9.1
- | • The following utilities are new or changed:
  - | – apt (Run the QShell apt command, the Java™ annotation processing tool)
  - | – pack200 (Run the QShell pack200 command, the Java archive packing tool)
  - | – unpack200 (Run the QShell unpack200 command, the Java archive unpacking tool)
- | • New or changed i5/OS PASE runtime function:
  - | – \_OPEN\_CCSD (Open with CCSID for i5/OS PASE)
- | • New locales were added.
- | • New examples were added.

## How to see what’s new or changed

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

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

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

### Related concepts

“Install AIX compilers on i5/OS PASE” on page 10

You can follow the steps in this topic to install AIX compilers on i5/OS PASE.

### Related information

i5/OS PASE shells and utilities

Runtime functions for use by i5/OS PASE programs

i5/OS PASE locales

---

## Printable PDF

You can use this to view and print a PDF of this information.


To view or download the PDF version of this document, select i5/OS PASE (about 645 KB).

## Saving PDF files

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

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

## Downloading Adobe Reader

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

---

## Get started with i5/OS PASE

i5/OS Portable Application Solutions Environment (i5/OS PASE) enables you to run many of your AIX application binaries on i5/OS with little or no change, and effectively expands your platform solution portfolio.

Cross-platform application development and deployment are crucial components of any effective business computing environment. Equally important are the ease of use and integration of functions that your system offers: the hallmarks of the iSeries. As your business moves into an increasingly open computing environment, you are likely to find that achieving these often divergent goals can be difficult, time-consuming, and expensive. For instance, you might want the benefit of a familiar application that runs on and makes use of the capabilities of the AIX operating system, but you do not want the added burden of managing both AIX and i5/OS operating systems.

This is where i5/OS Portable Application Solutions Environment (i5/OS PASE) helps. i5/OS PASE lets you run many of your AIX application binaries on i5/OS with little or no change, and effectively expands your platform solution portfolio.

## What is i5/OS PASE?

i5/OS PASE is an integrated runtime environment for AIX applications running on i5/OS.

It supports the application binary interface (ABI) of AIX, and provides a broad subset of the support provided by AIX shared libraries, shells, and utilities. i5/OS PASE supports the direct processing of IBM PowerPC<sup>®</sup> machine instructions, so it does not have the drawbacks of an environment that only emulates the machine instructions.

i5/OS PASE applications:

- Can be written in C, C++, Fortran, or PowerPC assembler
- Use the same binary executable format as AIX PowerPC applications
- Run in an i5/OS job
- Use i5/OS system functions, such as file systems, security, and sockets



Keep in mind that i5/OS PASE is not a UNIX<sup>®</sup> operating system on i5/OS. i5/OS PASE is designed to run AIX programs on i5/OS with little or no change. Programs from any other environment, such as AIX or Linux, need to be written such that they can be compiled on AIX as the first step toward running in i5/OS PASE.

The i5/OS PASE integrated run time runs on the Licensed Internal Code kernel on the iSeries server. The system provides integration of many common i5/OS functions across i5/OS PASE and other runtime environments (including ILE and Java). i5/OS PASE implements a broad subset of AIX system calls. System support for i5/OS PASE enforces system security and integrity by controlling what memory an i5/OS PASE program can access and restricting it to use only unprivileged machine instructions.

## **Rapid application deployment with minimal effort**

In many cases, your AIX programs can run in i5/OS PASE with little or no change. The level of AIX programming skills you need varies depending on the design of your AIX program. In addition, by providing additional i5/OS application integration in your program design (for instance, with CL commands), you can minimize configuration concerns for your application users.

i5/OS PASE adds another porting option for solution developers who want to share in the success of the i5/OS marketplace. By providing a means to cut porting time significantly, i5/OS PASE can improve the time to market and return on investment for solutions developers.

## **A broad subset of AIX technology on i5/OS**

i5/OS PASE implements an application run time that is based on a broad subset of AIX technology, including:

- Standard C and C++ run time (both threadsafe and non-threadsafe)
- Fortran run time (both threadsafe and non-threadsafe)
- pthreads threading package
- iconv services for data conversion
- Berkeley Software Distributions (BSD) equivalent support
- X Window System client support with Motif widget set
- Pseudo terminal (PTY) support

Applications are developed and compiled on an AIX workstation running a level of AIX that is compatible with a level supported by i5/OS PASE, and then run on i5/OS.

Alternatively, you can install one of the supported compiler products in the i5/OS PASE environment to develop, compile, build, and run your applications completely within i5/OS PASE.

i5/OS PASE also includes the Korn, Bourne, and C shells and nearly 200 utilities that provide a powerful scripting environment.

i5/OS PASE uses IBM investment in a common processor technology for the AIX and i5/OS operating systems. The PowerPC processor switches from i5/OS mode into AIX mode to run an application in the i5/OS PASE run time.

Applications running in i5/OS PASE are integrated with the i5/OS integrated file system and DB2 Universal Database<sup>™</sup> for iSeries. They can call (and be called by) Java and Integrated Language Environment<sup>®</sup> (ILE) applications. In general, they can take advantage of all aspects of the i5/OS operations environment, such as security, message handling, communication, and backup and recovery. At the same time, they take advantage of application interfaces that are derived from AIX interfaces.

### **Related reference**

“Compile your AIX source” on page 8

You can install one of the AIX compiler products that support installation in i5/OS PASE to compile your programs in the i5/OS PASE environment.

#### **Related information**

i5/OS PASE shells and utilities

## **When is i5/OS PASE a useful option for application development?**

You can use API analysis to determine whether an application is suitable for i5/OS PASE. i5/OS PASE is not the best solution under some circumstances.

i5/OS PASE provides considerable flexibility when you are deciding how to port your AIX applications to the iSeries server. Of course, i5/OS PASE is only one option of several from which you can choose.

### **API analysis**

Your starting point for determining whether an application is suitable for i5/OS PASE is an analysis of the application: the APIs, libraries, and utilities that it uses and how effectively it will run on i5/OS. The IBM Virtual Innovation Center for Hardware offers help in this area with the API Analysis Tool, a free porting assessment tool that analyzes your application and describes potential stumbling blocks. For more information about how the analysis tool fits into the procedures for porting applications to i5/OS PASE, see the “Prepare programs to run in i5/OS PASE” on page 7 topic.

### **Characteristics of a potential i5/OS PASE application**

Here are some useful guidelines that you might consider when making the decision whether to use i5/OS PASE:

- **Is the AIX application highly compute-intensive?**  
i5/OS PASE provides a good environment for running computation-intensive applications on iSeries servers by providing highly optimized math libraries.
- **Does the application rely heavily on functions that are supported only in i5/OS PASE (or only partially supported in ILE), such as fork(), X Window System, or pseudo-terminal (PTY) support?**  
i5/OS PASE provides support for fork() and exec(), which do not currently exist on the i5/OS system (except through spawn(), which incorporates the fork() function with the exec() function).
- **Does the application use a complicated AIX system-based build process or testing environment?**  
i5/OS PASE lets you use AIX system-based build processes, which are especially useful when you have an existing, complicated process that is not readily transferred onto a new operating system.
- **Does the application have dependencies on an ASCII character set?**  
i5/OS PASE provides good support for applications with these needs.
- **Does the application do a lot of pointer manipulation, or does it convert (cast) integers to pointers?**  
i5/OS PASE supports both 32- and 64-bit AIX addressing models with low performance cost and the ability to convert integers to pointers.

### **When i5/OS PASE might not be the best solution**

i5/OS PASE is generally not a good choice for code that provides a large number of callable interfaces that must be called from ILE and that has any of the following characteristics:

- Code that needs higher performance call and return than provided by either starting or ending i5/OS PASE on each call or by calling an i5/OS PASE procedure in an already-active i5/OS PASE program (using the Qp2CallPase API).
- Code that needs to share memory or namespace between an ILE caller and the library code. An i5/OS PASE program does not implicitly share memory or namespace with ILE code that called it. (However, ILE code that is called from i5/OS PASE can share or use i5/OS PASE memory.)

## Related information

API Analysis Tool

IBM Virtual Innovation Center for Hardware

## Install i5/OS PASE

You can follow the instructions in this topic to install i5/OS PASE on your server.

i5/OS PASE is available free of charge on all iSeries servers. It is recommended that you install i5/OS PASE; some system software, such as the enhanced Domain Name System (DNS) server and the ILE C++ compiler, requires i5/OS PASE support.

To install i5/OS PASE on your server, complete the following steps:

1. On an i5/OS command line, enter GO LICPGM.
2. Select 11 (Install licensed program).
3. Select option 33 (5722SS1 - Portable Application Solutions Environment).
4. **Optional:** Install additional locales.

The i5/OS PASE product installs only the locale objects that are associated with the language features that you have installed on i5/OS. If you need locales that are not included with the language features on your server, you need to order and install additional i5/OS language features. See i5/OS PASE Globalization and i5/OS PASE locales for more information.

### Licensing note for software developers who are porting an application to i5/OS PASE:

i5/OS PASE provides a subset of the AIX runtime libraries on the i5/OS system. The i5/OS license authorizes you to use any library code shipped with i5/OS. This license does not imply a license to AIX libraries that were not shipped with i5/OS PASE. All AIX products are separately licensed by IBM.

As you begin porting your own applications to i5/OS PASE, you might find that your application has dependencies on AIX libraries that were not shipped with i5/OS PASE. Before porting these libraries to the i5/OS system, you should determine which software product provided those libraries and examine the terms and conditions of the license agreement for that software product. It might be necessary to work with IBM or a third party to port additional middleware dependencies to the i5/OS system. You should investigate every licensing agreement involved with the code you are porting before you start porting. If you need to find out about license agreements in place against libraries that you believe belong to IBM, contact your IBM marketing representative, one of the IBM porting centers, the Custom Technology Center in Rochester, or PartnerWorld<sup>®</sup> for Developers.

#### Related concepts

“Globalization” on page 48

Because the i5/OS PASE run time is based on the AIX run time, i5/OS PASE programs can use the same rich set of programming interfaces for locales, character string manipulation, date and time services, message catalogs, and character encoding conversions supported on AIX.

#### Related information

i5/OS PASE locales

---

## Plan for i5/OS PASE

When you begin to work with i5/OS PASE, the points listed in this topic might be helpful to you.

i5/OS PASE provides an AIX runtime environment on i5/OS that lets you port your AIX applications to the iSeries server with minimal effort. In fact, many AIX programs run in i5/OS PASE with no change. This is because i5/OS PASE supplies many of the same shared libraries that are available on AIX, and it provides a broad subset of AIX utilities that run directly on the iSeries PowerPC processor in the same way that they run on the pSeries<sup>®</sup> AIX PowerPC processor.

Some points to keep in mind as you begin to work with i5/OS PASE:

- **There is a correlation between the target release of an AIX binary and the release of i5/OS PASE where the binary will run.**

If you compile your i5/OS PASE applications on AIX, the application binary created on AIX needs to be compatible with the version of i5/OS PASE that you want the application to run in. The following table shows which AIX binary versions are compatible with different versions of i5/OS PASE. For example, a 32-bit application created for AIX release 5.1 will run on i5/OS PASE V5R4, V5R3, or OS/400® PASE V5R2, but not on OS/400 PASE V5R1. Similarly, a 64-bit application created for AIX release 4.3 will run on OS/400 PASE V5R1, but not on i5/OS PASE V5R4, V5R3, or OS/400 PASE V5R2.

| AIX release         | OS/400 V5R1 | OS/400 V5R2 | i5/OS V5R3 | i5/OS V5R4 |
|---------------------|-------------|-------------|------------|------------|
| 4.3 (32-bit)        | X           | X           | X          | X          |
| 4.3 (64-bit)        | X           | -           | -          | -          |
| 5.1 (32- or 64-bit) | -           | X           | X          | X          |
| 5.2 (32- or 64-bit) | -           | -           | X          | X          |
| 5.3 (32- or 64-bit) | -           | -           | -          | X          |

- **i5/OS PASE does not provide the AIX kernel on i5/OS.**

Instead, any low-level system functions that are needed by a shared library are routed to the i5/OS kernel or to the integrated i5/OS functions. In this regard, i5/OS PASE bridges the gap across the AIX and i5/OS platforms: your code uses the same syntax for the APIs in the shared libraries as you can find on AIX, but your i5/OS PASE program runs within an i5/OS job and is managed by i5/OS just like any other i5/OS job.

- **In most cases, the APIs you call in i5/OS PASE behave in exactly the same manner as they do on AIX.**

Some APIs, however, might behave differently in i5/OS PASE, or might not be supported in i5/OS PASE. Because of this, your plan for preparing i5/OS PASE programs should begin with a thorough code analysis using the API Analysis Tool. This tool gives you a comprehensive summary of the types of program modifications you need to consider in porting your AIX application to i5/OS PASE.

- **Consider some of the differences that exist between the AIX and i5/OS platforms:**

- AIX is generally case-sensitive, but certain i5/OS file systems are not.
- AIX generally uses ASCII for data encoding, but i5/OS generally uses Extended Binary Coded Decimal Interchange Code. This will be a consideration if you want to manage the details of calling ILE code from your i5/OS PASE program. For example, you must explicitly code i5/OS PASE programs to handle character encoding conversions on strings when you make calls from i5/OS PASE to arbitrary ILE procedures. i5/OS PASE runtime support includes the `iconv_open()`, `iconv()`, and `iconv_close()` functions for character encoding conversion.

**Note:** i5/OS PASE and ILE have independent implementations of `iconv()` interfaces, each with its own translation tables. The translations supported by i5/OS PASE `iconv()` support can be modified and extended by users because they are stored as byte stream files in the integrated file system.

- AIX applications expect that lines (for example, in files and shell scripts) will end with a line feed (LF), but personal computer (PC) software and i5/OS software typically end lines with a carriage return and line feed (CRLF).
- Some of the scripts and programs you use on AIX might use hardcoded paths to standard utilities, and you might need to modify the path to reflect the paths you will be using in i5/OS PASE. See *Analyze your program's compatibility with i5/OS PASE* for more information.

i5/OS PASE automatically handles some of these issues. For example, when you use the i5/OS PASE runtime service that the system provides (including any system call or runtime function in a shared

library shipped with i5/OS option 33), i5/OS PASE performs ASCII-to-EBCDIC conversions as needed, although generally no conversions are done for data that is read or written to a file descriptor (byte stream file or socket).

You can use other low-level functions, such as `_ILECALL`, to extend the functionality of your i5/OS PASE program with calls to ILE functions and APIs, but as mentioned above you might need to handle data conversion. Also, coding these extensions into your program requires the use of additional header and export files.

#### **Related concepts**

“Analyze your program’s compatibility with i5/OS PASE”

The first step in an assessment of the portability of a C application to the iSeries server involves the analysis of the interfaces that are used in your application.

---

## **Prepare programs to run in i5/OS PASE**

The steps you need to take to prepare AIX programs that run effectively on i5/OS vary with the nature of your program and your actual needs for i5/OS system-unique interfaces and functions.

If you are attempting to port an application to i5/OS PASE, you must first ensure that the application will compile using an AIX compiler. In some cases, you need to modify your program to achieve this requirement.

## **Analyze your program’s compatibility with i5/OS PASE**

The first step in an assessment of the portability of a C application to the iSeries server involves the analysis of the interfaces that are used in your application.

This API analysis identifies those interfaces that are used within the application that are not industry standard and not supported on i5/OS. It also identifies the interfaces that are compliant with industry standard but supported differently because of the different architecture of i5/OS compared to AIX or Linux machines.

The API Analysis Tool consists of front-end and back-end processes. The front-end process scans the compiled application to extract the interfaces (external functions and data) that are used by the application, and generates a list of all those interfaces. The back-end process takes this list of interfaces as input and compares the interfaces with a database of typical system APIs and their support.

The front-end process of the API analysis tool is a shell script. It uses the `nm` or `dump` command to find symbol information from the external symbol table of the application.

Binaries that have been stripped of symbols can contain enough dynamic binding information for the tool to analyze. Statically bound binaries remove the library interfaces from the analysis but still expose system call dependencies for analysis.

## **Additional analysis to perform before you compile**

In addition to the information you gather from the API analysis tool, you should also gather the following information:

- Obtain a list of libraries used by your application

The analysis tool gives you feedback on some of the standard APIs that your application uses, but it does not look for many common API sets. A library analysis helps identify some of the middleware APIs that your application uses. You can run the following command against each of your commands and shared objects to get a list of libraries required by your application:

```
dump -H binary_name
```

- Check your code for hardcoded path names

If you run programs that change credentials or want your programs or scripts to run even when the i5/OS PASE environment variable `PASE_EXEC_QOPENSYS=N`, you might need to change hardcoded path names.

Because `/usr/bin/ksh` is an absolute path (starting at the root), if it is not found or if it is not a byte stream file, i5/OS PASE searches the /QOpenSys file system for path name `/QOpenSys/usr/bin/ksh`. QShell utility programs are not byte stream files, so i5/OS PASE searches the /QOpenSys file system even when the original (absolute) path is a symbolic link to a QShell utility program, such as `/usr/bin/sh`.

#### **Related concepts**

“Plan for i5/OS PASE” on page 5

When you begin to work with i5/OS PASE, the points listed in this topic might be helpful to you.

#### **Related information**

API Analysis Tool

## **Compile your AIX source**

You can install one of the AIX compiler products that support installation in i5/OS PASE to compile your programs in the i5/OS PASE environment.

When your program uses AIX interfaces only, you compile with any required AIX headers and link with AIX libraries to prepare binaries for i5/OS PASE. Keep in mind that i5/OS PASE does not support applications that are statically bound with AIX system-supplied shared libraries.

i5/OS PASE programs are structurally identical to AIX programs for PowerPC.

i5/OS PASE (option 33 of the operating system) does not include a compiler. You use an AIX system to compile i5/OS PASE programs, or you can optionally install one of the AIX compiler products that support installation in i5/OS PASE to compile your programs in the i5/OS PASE environment.

## **Use AIX compilers on the pSeries server**

You can build i5/OS PASE programs using any AIX compiler and linker that generate output that is compatible with the AIX ABI for PowerPC. i5/OS PASE provides instruction emulation support for binaries that use POWER™ architecture instructions that do not exist in PowerPC (except for IBM POWER instructions for cache management).

## **Use AIX compilers in i5/OS PASE**

i5/OS PASE supports the installation of the following separately available AIX compilers in the i5/OS PASE environment:

- | • IBM XL C/C++ Enterprise Edition for AIX, V7.0 (5724-I11)
- | • IBM XL C Enterprise Edition for AIX, V7.0 (5724-I10)
- | • IBM XL Fortran Enterprise Edition for AIX, V9.1 (5724-I08)

Using these products, you can develop, compile, build, and run your AIX applications entirely within the i5/OS PASE environment on your iSeries server.

## **Development tools**

Many development tools that you use on AIX (for example, `ld`, `ar`, `make`, `yacc`) are included with i5/OS PASE. Many AIX tools from other sources (for instance, the open-source tool `gcc`) can also work in i5/OS PASE.



The iSeries Tools for Developers PRPQ (5799-PTL) also contains a wide array of tools to aid in the development, building, and porting of iSeries applications. For more information about this PRPQ, see the IBM Virtual Innovation Center for Hardware Web site.

## Compiler notes for handling of pointers

- The `xlc` compiler provides limited support for 16-byte alignment (for type long double) by using the combination of `-qlngdbl128` and `-qalign=natural`. Type *ILEpointer* requires these compiler options to ensure that machine interface (MI) pointers are 16-byte aligned within structures. Using option `-qldb128` forces type long double to be a 128-bit type that requires use of `libc128.a` to handle operations like `printf` for long double fields.

An easy way to get option `-qlngdbl128` and link with `libc128.a` is to use the `xlc128` command instead of the `xlc` command.

- The `xlc/xlc` compiler currently does not provide a way to force 16-byte alignment for static or automatic variables. The compiler only guarantees relative alignment for 128-bit long double fields within structures. The i5/OS PASE version of `malloc` always provides 16-byte aligned storage, and you can arrange 16-byte alignment of stack storage.
- Header file `as400_types.h` also relies on type `long long` to be a 64-bit integer. `xlc` compiler option `-qlonglong` ensures this geometry (which is not the default for all commands that run the `xlc` compiler).

## Examples

The following examples are intended for use when you are compiling your i5/OS PASE programs on an AIX system. If you are using a compiler installed in i5/OS PASE to compile your programs, you do not need to specify compiler options for the locations of i5/OS system-unique header files or i5/OS system-unique exports because these files will be found in their default path locations of `/usr/include/` and `/usr/lib/` on an i5/OS system.

### Example 1

The following command on an AIX system creates an i5/OS PASE program named `testpgm` that can use i5/OS system-unique interfaces exported by `libc.a`:

```
xlc -o testpgm -qldb128 -qlonglong -qalign=natural
      -bI:/mydir/as400_libc.exp testpgm.c
```

This example assumes that the i5/OS system-unique header files are copied to the AIX directory `/usr/include` and that the i5/OS system-unique exports files are copied to the AIX directory `/mydir`.

### Example 2

The following example assumes i5/OS system-unique headers and export files are in `/pase/lib`:

```
xlc -o as400_test -qldb128 -qlonglong -qalign=natural -H16
      -l c128
      -I /pase/lib
      -bI:/pase/lib/as400_libc.exp as400_test.c
```

### Example 3

The following example builds the same program as example 2 with the same options; however, the `xlc_r` command is used for a multithreaded program to ensure that the compiled application links with threadsafe runtime libraries:

```
xlc_r -o as400_test -qldb128 -qlonglong -qalign=natural -H16
      -l c128
      -I /pase/lib
      -bI:/pase/lib/as400_libc.exp as400_test.c
```

In the examples, if you are using i5/OS PASE support for IBM DB2 Universal Database (UDB) for iSeries call level interfaces (CLI), you also need to specify `-bI:/pase/include/libdb400.exp` on your build command.

The `-bI` directive tells the compiler to pass the parameter to the `ld` command. The directive specifies an export file containing exported symbols from a library to be imported by the application.

#### **Related concepts**

“What is i5/OS PASE?” on page 2

i5/OS PASE is an integrated runtime environment for AIX applications running on i5/OS.

#### **Related information**

i5/OS PASE shells and utilities

## **Install AIX compilers on i5/OS PASE**

You can follow the steps in this topic to install AIX compilers on i5/OS PASE.

You can install either of the following separately available AIX compilers in the i5/OS PASE environment:

- | • IBM XL C/C++ Enterprise Edition for AIX, V7.0 (5724-I11)
- | • IBM XL C Enterprise Edition for AIX, V7.0 (5724-I10)
- | • IBM XL Fortran Enterprise Edition for AIX, V9.1 (5724-I08)

These products let you develop, compile, build, and run your AIX applications entirely within the i5/OS PASE environment on your iSeries server.

#### **Related concepts**

“What’s new for V5R4” on page 1

This page highlights the changes to the i5/OS PASE in this release.

#### **Related information**

XL C/C++ Enterprise Edition for AIX

XL C Enterprise Edition for AIX

### **Install the AIX compilers:**

i5/OS PASE does not support the AIX `smit` or `installp` utilities typically used to install applications on an AIX system. Installation of the XL C/C++ Enterprise Edition for AIX V7.0 or XL C Enterprise Edition for AIX products is accomplished through a “non-default installation” script included on the respective compiler’s installation media.

- | The following steps will install the XL C/C++ Enterprise Edition for AIX V7.0 or XL C Enterprise Edition for AIX product on iSeries i5/OS PASE:
- | 1. Verify you have the necessary prerequisites. In addition to the compiler installation media, you also need the following programmes installed on your iSeries server to successfully install and use the compiler:
  - | • 5722SS1 Option 33 - i5/OS PASE itself
  - | • 5722SS1 Option 13 - System Openness Includes, containing the compiler header files found in the `/usr/include` integrated file system directory
  - | • Perl. The compiler installation scripts require Perl. Here are two ways to install Perl:
    - | – 5799PTL - iSeries Tools for Developers PRPQ. Perl (along with many other useful development tools) is included in the separately available iSeries Tools For Developers PRPQ.
    - | – <http://www.cpan.org/ports/#os400> - A Perl Port binary distribution for i5/OS PASE
- | 2. Insert the compiler product installation CD into the iSeries CDROM device.
- | 3. Sign on to i5/OS with a user profile that has `*ALLOBJ` authority. The compiler product files will be owned by this user profile.



- | 4. Start an interactive i5/OS PASE terminal session by entering this CL command: call qp2term
- | 5. Restore the appropriate compiler installation script by entering these commands:

| Compiler                                   | Command   |
|--|---|
| For XL C/C++ Enterprise Edition for AIX:   | cd /<br>restore -qf /QOPT/CDROM/USR/SYS/INST.IMA/VACPP.NDI ./usr/vacpp/bin/vacppndi |
| For XL C Enterprise Edition for AIX:       | cd /<br>restore -qf /QOPT/CDROM/USR/SYS/INST.IMA/VAC.NDI ./usr/vac/bin/vacndi       |
| For XL Fortran Enterprise Edition for AIX: | cd /<br>restore -qf /QOPT/CDROM/USR/SYS/INST.IMA/XLF.NDI ./usr/lpp/xlf/bin/xlfndi   |

- | 6. Run the installation script to install the compiler. The destination directory for the compiler is specified by the -b option in the command. The recommended directory names for the compilers are used in the commands in the following table. If you choose a different directory, note that the directory should be in the /QOpenSys tree (to allow for case-sensitive file names):

| Compiler   | Command   |
|--|---|
| For XL C/C++ Enterprise Edition for AIX:             | /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vacpp/bin/vacppndi -i -d /QOPT/CDROM/USR/SYS/INST.IMA -b /QOpenSys/xlc70 |
| For XL C Enterprise Edition for AIX:                 | /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vac/bin/vacndi -i -d /QOPT/CDROM/USR/SYS/INST.IMA -b /QOpenSys/xlc70     |
| For XL Fortran Enterprise Edition for AIX:           | /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/lpp/xlf/bin/xlfndi -i -d /QOPT/CDROM/USR/SYS/INST.IMA -b /QOpenSys/xlf91 |
| <b>Note:</b> Enter the commands as one long command. |   |

| The compiler is now installed for use in i5/OS PASE.

| The XL C/C++ Enterprise Edition for AIX compiler commands, such as xlc, can be found in directory /QOpenSys/xlc70/usr/vacpp/bin/. You might want to add this directory to your \$PATH environment variable.

| The XL C/C++ Enterprise Edition for AIX compiler documentation can be found in Adobe Acrobat format in directory /QOpenSys/xlc70/usr/vacpp/pdf/en\_US/.

| The XL C Enterprise Edition for AIX compiler commands, such as xlc and cc, can be found in directory /QOpenSys/xlc70/usr/vac/bin/. You might want to add this directory to your \$PATH environment variable.

| The XL C Enterprise Edition for AIX compiler documentation can be found in Adobe Acrobat format in directory /QOpenSys/xlc70/usr/vac/pdf/en\_US/.

| The XL Fortran for AIX compiler commands, such as xlf, can be found in directory /QOpenSys/xlf91/usr/bin/. You might want to add this directory to your \$PATH environment variable.

| The XL Fortran for AIX compiler documentation can be found in Adobe Acrobat format in directory /QOpenSys/xlf91/usr/share/man/info/en\_US/xlf/pdf/.

**PTF update instructions:**

Installation of program temporary fixes (PTFs) for the XL C/C++ Enterprise Edition for AIX V7.0 or XL C Enterprise Edition for AIX products is accomplished using the same "non-default installation" script that is used for the initial compiler installation.

Before installing the PTFs, you must have already installed the compilers using the previous steps in this topic.

- | The following steps will install PTFs for the XL C/C++ Enterprise Edition for AIX V7.0 or XL C Enterprise Edition for AIX product on iSeries i5/OS PASE.
- | 1. Obtain the PTF package files to be installed. You can download compressed TAR images of the compiler PTF packages from the support downloads section of the XL C/C++ Enterprise Edition Web site.
- | 2. Uncompress and then untar the PTF package files. If you have downloaded the compressed TAR images to the /QOpenSys/vacptf/ directory, you can use these commands from a QP2TERM command line to do this:
 

```
| cd /QOpenSys/ptf
| uncompress <filename.tar.Z>
| tar -xvf <filename.tar>
```
- | 3. Create a file containing a list of the PTF packages to be installed. Use these commands from a QP2TERM command line to do this:
 

```
| cd /QOpenSys/ptf
| ls *.bff > ptflist.txt
```
- | 4. Run the installation script to install the PTFs. Based on the compiler you are updating, enter one of the following commands from the QP2TERM command line:

| Compiler                                   | Command  |
|--|--|
| For XL C/C++ Enterprise Edition for AIX:   | /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vacpp/bin/vacppndi -d /QOpenSys/ptf -b /QOpenSys/xlc70 -u /QOpenSys/ptf/ptflist.txt |
| For XL C Enterprise Edition for AIX:       | /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vac/bin/vacndi -d /QOpenSys/ptf -b /QOpenSys/xlc70 -u /QOpenSys/ptf/ptflist.txt     |
| For XL Fortran Enterprise Edition for AIX: | /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/lpp/xlf/bin/xlfndi -d /QOpenSys/ptf -b /QOpenSys/xlf91 -u /QOpenSys/ptf/ptflist.txt |

**Note:** Enter the commands as one long command.

The installation script creates a compressed TAR backup of the compiler files that existed before the PTF update. If you use the directories as shown in these instructions, this file will be named /QOpenSys/xlc70.backup.tar.Z or /QOpenSys/xlf91.backup.tar.Z. If a problem is encountered with the installation of the PTF update or the PTF update itself, you can restore from this backup to uninstall the PTF update.

**Related information**

XL C/C++ Enterprise Edition for AIX

## Copy the i5/OS PASE program to your iSeries server

Copy AIX binaries that you want to run in i5/OS PASE into the integrated file system.

All of the file systems that are available in the integrated file system are available within i5/OS PASE.

When you move files across operating systems, be aware of your application’s sensitivity to mixed case and of the difference between line-terminating characters that AIX uses and those that i5/OS uses. These differences can create problems for you.

You can transfer your i5/OS PASE program and related files to and from your iSeries server by using File Transfer Protocol (FTP), Server Message Block (SMB), or remote file systems.

**Related reference**

“Copy header files” on page 16

You can follow the instructions in this topic to copy header files from your iSeries server to an AIX machine.

“Copy export files” on page 17

You can follow the instructions in this topic to copy the export files from your iSeries server to an AIX directory.

#### **Related information**

Integrated file system

### **Case sensitivity**

If your application is sensitive to mixed case, move it into the /QOpenSys file system, or into a user-defined file system that has been created as case-sensitive.

The interfaces of operating systems, such as AIX and Linux, generally differentiate between uppercase and lowercase letters. On i5/OS, that is not always the case. You should be aware of several situations in particular where case sensitivity might cause complications with existing code.

Case sensitivity on a directory or file basis depends on the file system you are using on i5/OS. The /QOpenSys file system is case sensitive, and you can create a user-defined file system (UDFS) that is case sensitive.

### **Examples**

The following examples are problems stemming from case sensitivity that you might encounter.

#### **Example 1**

In this example, the shell does a character comparison of the generic name prefix against what is returned by `readdir()`. However, the QSYS.LIB file system returns directory entries in uppercase, so none of the entries matches the lowercase generic name prefix.

```
$ ls -d /qsys.lib/v4r5m0.lib/qwobj*
/qsys.lib/v4r5m0.lib/qwobj* not found
$ ls -d /qsys.lib/v4r5m0.lib/QW0BJ*
/qsys.lib/v4r5m0.lib/QW0BJ.FILE
```

#### **Example 2**

This example is similar to the first example except that, in this case, the `find` utility is doing the comparison, and not the shell.

```
$ find /qsys.lib/v4r5m0.lib/ -name 'qwobj*' -print
$ find /qsys.lib/v4r5m0.lib/ -name 'QW0BJ*' -print
/qsys.lib/v4r5m0.lib/QW0BJ.FILE
```

#### **Example 3**

The `ps` utility expects user names to be case-sensitive and therefore does not recognize a match between the uppercase name specified for the `-u` option and lowercase names returned by the i5/OS PASE runtime function `getpwuid()`:

```
$ ps -uTIMMS -f
UID PID PPID C STIME TTY TIME CMD
$ ps -utimms -f
UID PID PPID C STIME TTY TIME CMD
timms 617 570 0 10:54:00 - 0:00 /QOpenSys/usr/bin/-sh -i
timms 660 617 0 11:14:56 - 0:00 ps -utimms -f
```

#### **Related information**

## Line terminating characters in integrated file system files

AIX and i5/OS use different line terminating characters in text files (for example, in files and shell scripts).

The AIX applications that are the source for your i5/OS PASE programs expect that lines (for example, in files and shell scripts) will end with a line feed (LF). However, PC software and typical i5/OS software often ends lines with a carriage return and line feed (CRLF).

```
awk '{ gsub( /\r$/, "" ); print $0 }' < oldfile > newfile
```

## CRLF used with FTP

One example of where this difference can cause problems is when you use File Transfer Protocol (FTP) to transfer source files and shell scripts from AIX to the iSeries. The FTP standard calls for data sent in text mode to use carriage return and line feed (CRLF) at the end of a line. On AIX, the FTP utility strips the carriage return (CR) when it processes an inbound file in text mode. i5/OS FTP always writes exactly what is presented in the data stream and always retains CRLF for text mode, which causes problems with the i5/OS PASE run time and utilities.

Where possible, use binary mode transfer from an AIX operating system to avoid this problem. Text files transferred from personal computers will, in most cases, have CRLF delimiting lines in the file.

Transferring the files first to AIX will correct the problem. The following command can be used as a means to remove the CR from files in the current directory:

```
awk '{ gsub( /\r$/, "" ); print $0 }' < oldfile > newfile
```

## CRLF used with iSeries and PC editors

You can also experience problems when you edit your files or shell scripts with editors on your iSeries server or with editors on your workstation (such as Windows<sup>®</sup> Notepad editor). These editors use CRLF as a new line separator, and not the LF that i5/OS PASE expects.

Numerous editors are available (for instance, the ez editor) that do not use CRLF as new line separators.

### Related information

IBM Virtual Innovation Center for Hardware

## Transfer files

You can transfer your i5/OS PASE program and related files to and from your iSeries server in any of the three methods explained in this topic.

- Copy programs using File Transfer Protocol
- Copy programs using Server Message Block
- Copy programs using remote file systems

## Copy programs using File Transfer Protocol

You can use the i5/OS File Transfer Protocol (FTP) daemon and client to transfer a file into or out of the i5/OS integrated file system. Transfer your files in binary mode. Use the FTP subcommand `binary` to set this mode.

You must use naming format 1 (the `NAMEFMT 1` subcommand of the i5/OS FTP command) when placing files into the integrated file system. This format allows the use of path names, and transfers the files into stream files. To enter into naming format 1, you can either:

- Change the directory using path names.

This automatically puts the session into name format 1. Using this method, the first directory is prefaced by a slash (/). For example:

```
cd /QOpenSys/usr/bin
```

- Use the FTP subcommand `quote site namefmt 1` for a remote client, or use `namefmt 1` as a local client.

## Copy programs using Server Message Block

i5/OS supports Server Message Block (SMB) client and server components. With NetServer™ configured and running, i5/OS PASE has access to SMB servers in the network through the /QNTC file system. On an AIX or Linux platform, a SAMBA server is required to provide the same service. Installing a configured and operational system, such as AIX, can make directories and files available to i5/OS PASE.

## Copy programs using remote file systems

i5/OS lets you mount Network File System (NFS) file systems to a mount point in the integrated file system file space. AIX supports NFS, as well as Distributed File System (DFS™) and Andrew File System (AFS®) (using DFS-to-NFS and AFS-to-NFS translators) so that these file systems can be exported and mounted by i5/OS. This, in turn, lets i5/OS PASE applications use these file systems. Security authorization is validated through the i5/OS user profile's user ID number and group ID number for the directory path or file being accessed. You will want to ensure that a user profile that is intended to be the same person across multiple platforms has the same user ID on all of the systems.

i5/OS is best used as an NFS server. In this case, you need to mount from your AIX system onto a directory in the i5/OS integrated file system, and AIX will write programs directly onto i5/OS when they build.

**Note:** i5/OS NFS is currently not supported in multithreaded applications.

### Related information

FTP

## Customize i5/OS PASE programs to use i5/OS functions

If you want your AIX application to take advantage of i5/OS functions that are not directly supported by system-supplied i5/OS PASE shared libraries, you need to perform some additional steps to prepare your application.

Complete the following steps to do the preparation:

1. Code your AIX application to call any required i5/OS PASE runtime functions that coordinate your access to the i5/OS system-unique functions.
2. If you are compiling your i5/OS PASE programs on an AIX system, perform the following steps before you compile your customized application:
  - a. Copy required i5/OS system-unique header files to your AIX system.
  - b. Copy required i5/OS system-unique export files to your AIX system.

### Related concepts

“Call i5/OS programs and procedures from your i5/OS PASE programs” on page 28  
i5/OS PASE provides methods for calling ILE procedures, Java programs, OPM programs, i5/OS APIs, and CL commands that give you integrated access to i5/OS functions.

“How i5/OS PASE programs interact with i5/OS” on page 39

As you customize your i5/OS PASE programs to use i5/OS functions, you need to consider the ways in which your program will interact with them.

### Related information

Runtime functions for use by i5/OS PASE programs

## Copy header files

You can follow the instructions in this topic to copy header files from your iSeries server to an AIX machine.

i5/OS PASE augments standard AIX run time with header files for i5/OS system-unique support. These are provided by i5/OS PASE and the i5/OS operating system.

## Copy the header files from your iSeries server to an AIX machine in the header file search path

You can copy the header file into the /usr/include AIX directory, or to any other directory on the header file search path for your compiler.

If you use a directory other than /usr/include, you can add it to the header file search path with the -I option on the AIX compiler command.

## Copy i5/OS PASE header files

The i5/OS PASE header files are located in the following i5/OS directory:

```
/QOpenSys/QIBM/ProdData/OS400/PASE/include
```

i5/OS PASE provides the following header files:

| Header file    | Explanation  |
|----------------|--|
| as400_protos.h | This header file provides miscellaneous i5/OS PASE system-unique functions to ILE.   |
| as400_types.h  | This header file declares unique i5/OS parameter types for calls to ILE.<br><br>This header file declares type ILEpointer for 16-byte machine interface (MI) pointers, which relies on type long double to be a 128-bit field.<br><br>Other types declared in as400_types.h rely on type long long to be a 64-bit integer. AIX compilers must be run with options -q1ngdb1128, -qalign=natural, and -q1onglong to ensure proper size and alignment of types declared in as400_types.h. |
| os400msg.h     | This header file declares the functions to send and receive i5/OS messages.  |

## Copy i5/OS header files

If you plan to access other i5/OS functions in your i5/OS PASE application, you might find it helpful to copy to your development machine the header files for the i5/OS functions that you are using. Note that generally you cannot run an i5/OS program or procedure directly from an i5/OS PASE application. See Call i5/OS programs and procedures from your i5/OS PASE programs for more information.

i5/OS system-provided header files are located in the /QIBM/include directory:

If your application needs any of the i5/OS API header files, you must first convert them from EBCDIC to ASCII before you copy the converted files to an AIX directory.

One way to convert an EBCDIC text file to ASCII is to use the i5/OS PASE Rfile utility.

The following example uses the i5/OS PASE Rfile utility to read i5/OS header file /QIBM/include/qusec.h, convert the data to the i5/OS PASE coded character set identifier (CCSID), strip trailing blanks from each line, and then write the result into byte stream file ascii\_qusec.h:

```
Rfile -r /QIBM/include/qusec.h > ascii_qusec.h
```

**Related concepts**

“Database” on page 40

i5/OS PASE supports the DB2® UDB for iSeries Call Level Interface (CLI). DB2 CLI on AIX and i5/OS are not proper subsets of each other, so there are minor differences in a few interfaces, and some APIs in one implementation might not exist in another.

“Call i5/OS programs and procedures from your i5/OS PASE programs” on page 28

i5/OS PASE provides methods for calling ILE procedures, Java programs, OPM programs, i5/OS APIs, and CL commands that give you integrated access to i5/OS functions.

**Related tasks**

“Call ILE procedures” on page 28

You can follow the instructions in this topic to prepare and call ILE procedures from your i5/OS PASE programs.

**Related reference**

“Copy the i5/OS PASE program to your iSeries server” on page 12

Copy AIX binaries that you want to run in i5/OS PASE into the integrated file system.

**Copy export files**

You can follow the instructions in this topic to copy the export files from your iSeries server to an AIX directory.

The export files, located in the following i5/OS directory, are the recommended way to build your applications that require access to i5/OS system-specific functions:

/QOpenSys/QIBM/ProdData/OS400/PASE/lib

You can copy these files to any AIX directory. Use the -bI: option on the AIX ld command (or compiler command) to define symbols not found in the shared libraries on the AIX system.

i5/OS PASE provides the following export files:

| Export file    | Function  |
|----------------|---|
| as400_libc.exp | This file is the export file for i5/OS system-unique functions in libc.a<br><br>The as400_libc.exp file defines all the exports from the i5/OS PASE version of libc.a that are not exported by the AIX versions of those libraries. |
| libdb400.exp   | This file is the export file for i5/OS database functions<br><br>The libdb400.exp file defines the exports from the i5/OS PASE libdb400.a library (DB2 UDB for iSeries Call Level Interfaces (CLI) support).                        |

**Related concepts**

“Database” on page 40

i5/OS PASE supports the DB2 UDB for iSeries Call Level Interface (CLI). DB2 CLI on AIX and i5/OS are not proper subsets of each other, so there are minor differences in a few interfaces, and some APIs in one implementation might not exist in another.

**Related reference**

“Copy the i5/OS PASE program to your iSeries server” on page 12

Copy AIX binaries that you want to run in i5/OS PASE into the integrated file system.

**i5/OS PASE APIs for accessing i5/OS functions**

i5/OS PASE provides a number of APIs for accessing ILE code and other i5/OS functions. Which ones you use depends on how much preparation and structure building you want to do yourself as opposed to how much you want the compiler to do for you.

**Related information**



## Use i5/OS PASE programs in the i5/OS environment

Your i5/OS PASE program can call other i5/OS programs running in your job, and other i5/OS programs can call procedures in your i5/OS PASE program.

## Run i5/OS PASE programs and procedures

You can read this topic for information and examples about starting an i5/OS PASE program in a job, and calling i5/OS PASE procedures from your ILE programs.

You can run your i5/OS PASE program in any of several ways:

- Within an i5/OS job
- From an i5/OS PASE interactive shell environment
- As a called program from an ILE procedure

**Note:** When you run an i5/OS PASE program on i5/OS, keep in mind that the i5/OS PASE environment variables are independent of ILE environment variables. Setting a variable in one environment has no effect on the other environment.

## ILE procedures that let you work with i5/OS PASE programs

i5/OS PASE provides a number of ILE procedure APIs that allow your ILE code to access i5/OS PASE services (without special programming in your i5/OS PASE program):

- Qp2dlclose
- Qp2dlerror
- Qp2dlopen
- Qp2dlsym
- Qp2ernop
- Qp2free
- Qp2jobCCSID
- Qp2malloc
- Qp2paseCCSID
- Qp2ptrsize

## Attach to ILE threads

You can call a procedure in an i5/OS PASE program from ILE code that runs in a thread that was not created by i5/OS PASE (for example, a Java thread or a thread created by ILE pthread\_create). Qp2CallPase automatically attaches the ILE thread to i5/OS PASE (creating corresponding i5/OS PASE pthread structures), but only if the i5/OS PASE environment variable PASE\_THREAD\_ATTACH was set to Y when the i5/OS PASE program started.

## Return results from i5/OS PASE to i5/OS programs

Using the i5/OS \_RETURN() function, you can call an i5/OS PASE program and return results without ending the i5/OS PASE environment. This allows you to start an i5/OS PASE program and then call procedures in that program (using Qp2CallPase) after the QP2SHELL2 (but not QP2SHELL) or Qp2RunPase API returns.

### Related concepts



“Work with environment variables” on page 27

i5/OS PASE environment variables are independent of ILE environment variables. Setting a variable in one environment has no effect on the other environment.

#### **Related information**

i5/OS PASE ILE Procedure APIs

`_RETURN()`--Return Without Exiting i5/OS PASE

### **Run an i5/OS PASE program with QP2SHELL()**

You use QP2SHELL or QP2SHELL2 programs to run an i5/OS PASE program from any i5/OS command line and within any high-level language program, batch job, or interactive job.

These programs run an i5/OS PASE program in the job that calls it. The name of the i5/OS PASE program is passed as a parameter on the program.

The QP2SHELL() program runs the i5/OS PASE program in a new activation group. The QP2SHELL2() program runs in the caller's activation group.

The following example runs the `ls` command from the i5/OS command line:

```
call qp2shell parm('/QOpenSys/bin/ls' '/')
```

If you pass values into QP2SHELL() using CL variables, the variables must be null-terminated. For example, you need to code the above example in the following way:

```
PGM DCL VAR(&CMD) TYPE(*CHAR) LEN(20) VALUE('/QOpenSys/bin/ls')
DCL VAR(&PARAM1) TYPE(*CHAR) LEN(10) VALUE('/')
DCL VAR(&NULL) TYPE(*CHAR) LEN(1) VALUE(X'00')

CHGVAR VAR(&CMD) VALUE(&CMD *TCAT &NULL)
CHGVAR VAR(&PARAM1) VALUE(&PARAM1 *TCAT &NULL)

CALL PGM(QP2SHELL) PARM(&CMD &PARAM1)
```

```
ENDIT:
ENDPGM
```

#### **Related information**

QP2SHELL() and QP2SHELL2()--Run an i5/OS PASE Shell Program

### **Run an i5/OS PASE program with QP2TERM()**

You use this i5/OS program to run an i5/OS PASE program in an interactive shell environment.

Start an i5/OS PASE interactive terminal session with the QP2TERM() program.

The following command writes the default Korn shell prompt (`/QOpenSys/usr/bin/sh`) to the screen:

```
call qp2term
```

From this prompt, you run an i5/OS PASE program in a separate batch job. QP2TERM() uses the interactive job to display output and to accept input for files `stdin`, `stdout`, and `stderr` in the batch job.

The Korn shell is the default, but you can optionally specify the path name of any i5/OS PASE program that you want to run, as well as any argument strings to pass to the program.

You can run any i5/OS PASE program and any of the utilities from the interactive session that you start with QP2TERM(); `stdout` and `stderr` are written and scrolled in the terminal screen.

#### **Related information**

QP2TERM()--Run an i5/OS PASE Terminal Session

i5/OS PASE shells and utilities

## Run an i5/OS PASE program from within i5/OS programs

You can follow the steps in this topic to call the Qp2CallPase() and Qp2CallPase2() ILE procedures from within other ILE procedures to start and run an i5/OS PASE program. An example follows.

Use the Qp2RunPase() API to run an i5/OS PASE program. You specify the program name, argument strings, and environment variables.

The Qp2RunPase() API runs an i5/OS PASE program in the job where it is called. It loads an i5/OS PASE program (including any necessary shared libraries) and then transfers control to the program.

This API gives you more control over how i5/OS PASE runs than QP2SHELL() and QP2TERM().

### Related information

Qp2RunPase()--Run an i5/OS PASE Program

### Example: Run an i5/OS PASE program from within i5/OS programs:

The examples illustrated in this topic show an ILE program that calls an i5/OS PASE program, and the i5/OS PASE program that is called by the ILE program.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 56.

### Example 1: An ILE program that calls an i5/OS PASE program

The following ILE program calls an i5/OS PASE program. Following this example is an example of the i5/OS PASE code that this program calls.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

/* include file for QP2RunPase(). */

#include <qp2user.h>

/*****
Sample:
A simple ILE C program to invoke an i5/OS
PASE program using QP2RunPase() and
passing one string parameter.
Example compilation:
  CRTCMOD MODULE(MYLIB/SAMPLEILE) SRCFILE(MYLIB/QCSRC)
  CRTPGM PGM(MYLIB/SAMPLEILE)
*****/

void main(int argc, char*argv[])
{
  /* Path name of PASE program */
  char *PasePath = "/home/samplePASE";
  /* Return code from QP2RunPase() */
  int rc;
  /* The parameter to be passed to the
  i5/OS PASE program */
  char *PASE_parm = "My Parm";
  /* Argument list for i5/OS PASE program,
  which is a pointer to a list of pointers */
  char **arg_list;
  /* allocate the argument list */
  arg_list =(char**)malloc(3 * sizeof(*arg_list));
  /* set program name as first element. This is a UNIX convention */
```

```

arg_list[0] = PasePath;
/* set parameter as first element */
arg_list[1] = PASE_parm;
/* Last element of argument list must always be null */
arg_list[2] = 0;
/* Call i5/OS PASE program. */
rc = Qp2RunPase(PasePath, /* Path name */
  NULL,          /* Symbol for calling to ILE, not used in this sample */
  NULL,          /* Symbol data for ILE call, not used here */
  0,             /* Symbol data length for ILE call, not used here */
  819,          /* ASCII CCSID for i5/OS PASE */
  arg_list,      /* Arguments for i5/OS PASE program */
  NULL);        /* Environment variable list, not used in this sample */
}

```

## Example 2: The i5/OS PASE program that is called in the ILE program

The following i5/OS PASE program is called by the above ILE program.

```

#include <stdio.h>

/*****
Sample:
A simple i5/OS PASE Program called from
ILE using QP2RunPase() and accepting
one string parameter.
The ILE sample program expects this to be
located at /home/samplePASE. Compile on
AIX, then ftp to i5/OS.
To ftp use the commands:
> binary
> site namefmt 1
> put samplePASE /home/samplePASE
*****/

int main(int argc, char *argv[])
{
  /* Print out a greeting and the parameter passed in. Note argv[0] is the program
  name, so, argv[1] is the parameter */
  printf("Hello from i5/OS PASE program %s. Parameter value is \"%s\".\n", argv[0], argv[1]);

  return 0;
}

```

## Call an i5/OS PASE procedure from within i5/OS programs

You can call the Qp2CallPase() and Qp2CallPase2() ILE procedures from within other ILE procedures to run an i5/OS PASE program in a job where the i5/OS PASE environment is already running.

The Qp2RunPase() API initially starts and runs an i5/OS PASE program in a job. It returns an error if i5/OS PASE is already active in that job.

To call i5/OS PASE procedures in a job that is already running an i5/OS PASE program, you use the Qp2CallPase() and Qp2CallPase2() APIs.

### Related information

Qp2CallPase()--Call an i5/OS PASE Procedure

## Example 1: Call an i5/OS PASE procedure from within i5/OS programs:

The example in this topic shows an ILE program calling an i5/OS PASE procedure.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 56.

```

#include <stdio.h>
#include <qp2shell2.h>
#include <qp2user.h>
#define JOB_CC SID 0

int main(int argc, char *argv[])
{
    QP2_ptr64_t id;
    void *getpid_pase;
    const QP2_arg_type_t signature[] = { QP2_ARG_END };
    QP2_word_t result;

    /*
     * Call QP2SHELL2 to run the i5/OS PASE program
     * /usr/lib/start32, which starts i5/OS PASE in
     * 32-bit mode (and leaves it active on return)
     */
    QP2SHELL2("/usr/lib/start32");

    /*
     * Qp2dlopen opens the global name space (rather than
     * loading a new shared executable) when the first
     * argument is a null pointer. Qp2dlsym locates the
     * function descriptor for the i5/OS PASE getpid
     * subroutine (exported by shared library libc.a)
     */
    id = Qp2dlopen(NULL, QP2_RTLD_NOW, JOB_CC SID);
    getpid_pase = Qp2dlsym(id, "getpid", JOB_CC SID, NULL);

    /*
     * Call Qp2CallPase to run the i5/OS PASE getpid
     * function, and print the result. Use Qp2errnop
     * to find and print the i5/OS PASE errno if the
     * function result was -1
     */
    int rc = Qp2CallPase(getpid_pase,
                        NULL, // no argument list
                        signature,
                        QP2_RESULT_WORD,
                        &result);
    printf("i5/OS PASE getpid() = %i\n", result);
    if (result == -1)
        printf("i5/OS errno = %i\n", *Qp2errnop());

    /*
     * Close the Qp2dlopen instance, and then call
     * Qp2EndPase to end i5/OS PASE in this job
     */
    Qp2dlclose(id);
    Qp2EndPase();
    return 0;
}

```

### Example 2: An i5/OS ILE program that uses pointer arguments in a call to an i5/OS PASE procedure:

In this example, an i5/OS ILE program uses two different techniques to allocate and share memory storage with the i5/OS PASE procedure that it calls.

| **Note:** By using the following code examples, you agree to the terms of the “Code license and disclaimer information” on page 56.

```

| /* Name: ileMain.c
| *
| * Call an i5/OS PASE procedure from ILE
| *
| * This example uses the Qp2dlopen, Qp2dlsym, and Qp2CallPase2 ILE

```

```

| * functions to call an i5/OS PASE function passing in parameters
| *
| * Compile like so:
| *
| * CRTBNDC PGM(mylib/ilemain)
| *          SRCFILE(mylib/mysrcpf)
| *          TERASPACE(*YES *TSIFC)
| */
| #include <stdio.h>
| #include <stddef.h>
| #include <errno.h>
| #include <qp2user.h>
| /* Use EBCDIC default job CCSID in Qp2dlopen and Qp2dlsym calls */
| #define JOB_CCSID 0
|
| /* start i5/OS PASE in this process */
| void startPASE(void) {
|     /* start64 starts the 64 bit version of i5/OS PASE */
|     char *start64Path="/usr/lib/start64";
|     char *arg_list[2];
|
|     arg_list[0] = start64Path;
|     arg_list[1] = NULL;
|     Qp2RunPase(start64Path,
|               NULL,
|               NULL,
|               0,
|               819,
|               (char*)&arg_list,
|               NULL);
| }
|
| /* open a shared library */
| QP2_ptr64_t openlib(char * libname) {
|     QP2_ptr64_t id;
|     int * paseErrno;
|
|     /* Qp2dlopen dynamically loads the specified library returning an
|      * id value that can be used in calls to Qp2dlsym and Qp2dlcose */
|     id = Qp2dlopen(libname,
|                   (QP2_RTLD_NOW |
|                    QP2_RTLD_MEMBER ),
|                   JOB_CCSID);
|     if (id == 0) {
|         printf("Qp2dlopen failed. ILE errno=%i\n", errno);
|         if ((paseErrno=Qp2errnop()) != NULL)
|             printf("Qp2dlopen failed. i5/OS PASE errno=%i\n", *paseErrno);
|         printf("Qp2dlopen failed. Qp2dlerror = %s\n", Qp2dlerror());
|     }
|
|     return(id);
| }
|
| /* find an exported symbol */
| void * findsym(const QP2_ptr64_t id, const char * functionname) {
|     void * symbol;
|     int * paseErrno;
|
|     /* Qp2dlsym locates the function descriptor for the
|      * specified function */
|     symbol = Qp2dlsym(id, functionname, JOB_CCSID, NULL);
|     if (symbol == NULL) {
|         printf("Qp2dlsym failed. ILE errno = %i\n", errno);
|         if ((paseErrno=Qp2errnop()) != NULL)

```

```

|         printf("Qp2dlsym failed. i5/OS PASE errno=%i\n", *paseErrno);
|         printf("Qp2dlsym failed. Qp2dlerror = %s\n", Qp2dlerror());
|     }
|     return(symbol);
| }
|
| /* call i5/OS PASE procedure */
| int callPASE(const void * functionsymbol,
|             const void * arglist,
|             const QP2_arg_type_t * signature,
|             const QP2_result_type_t result_type,
|             void * buf,
|             const short buflen) {
|     int * paseErrno;
|     int rc;
|
|     /* Call Qp2CallPase2 to run the unction function */
|     rc = Qp2CallPase2(functionsymbol,
|                       arglist,
|                       signature,
|                       result_type,
|                       buf,
|                       buflen);
|
|     if (rc != 0) {
|         printf("Qp2CallPase failed. rc=%i, ILE errno=%i\n", rc, errno);
|         if ((paseErrno=Qp2errnop()) != NULL)
|             printf("Qp2CallPase failed. i5/OS PASE errno=%i\n", *paseErrno);
|         printf("Qp2CallPase failed. Qp2dlerror=%s\n", Qp2dlerror());
|     }
| }
|
| int main(int argc, char *argv[])
| {
|     /* we will call a function in i5/OS PASE named "paseFunction"
|     * the prototype for the function looks like this:
|     * int paseFunction(void * input, void * output ) */
|
|     /* "signature" is the argument signature for the PASE routine "paseFunction" */
|     const QP2_arg_type_t signature[] = {QP2_ARG_PTR64, QP2_ARG_PTR64, QP2_ARG_END};
|
|     /* "paseFunctionArglist" are the arguments for the PASE routine "paseFunction" */
|     struct {
|         QP2_ptr64_t inputPasePtr;
|         QP2_ptr64_t outputPasePtr;
|     } paseFunctionArglist;
|
|     /* "inputString" will be one of the arguments to the PASE routine
|     * "paseFunction" we will call
|     * This is the string "input" in ASCII */
|     const char inputString[] = {0x69, 0x6e, 0x70, 0x75, 0x74, 0x00};
|
|     /* "outputILEPtr" will be a pointer to storage malloc'd from PASE heap */
|     char * outputILEPtr;
|
|     /* "id" is the identifier for the library opened by Qp2dlopen */
|     QP2_ptr64_t id;
|
|     /* "paseFunction_ptr" is the pointer to the routine "paseFunction" in PASE */
|     void * paseFunction_ptr;
|
|     /* "inputAndResultBuffer" is the buffer of storage shared between ILE and PASE
|     * by Qp2CallPase2. This buffer contains space for the PASE function result */
|     struct {
|         QP2_dword_t result;
|         char inputValue[6];

```

```

} inputAndResultBuffer;

int rc;
int * paseErrno;

/* start i5/OS PASE in this process */
startPASE();

id = openlib("/home/joeuser/libpasefn.a(shr64.o)");

if (id !=0) {
    /* Locate the symbol for "paseFunction" */
    paseFunction_ptr = findsym(id, "paseFunction");

    if (paseFunction_ptr != NULL) {

        /* set input arguments for the call to paseFunction() */

        /* copy the inputString into the inputAndResultBuffer */
        strcpy(inputAndResultBuffer.inputValue, inputString);

        /* by setting inputPasePtr argument to the offset of the
         * inputValue by-address argument data in the
         * inputAndResultbuffer structure and OR'ing that with
         * QP2_ARG_PTR_TOSTACK QP2CallPase2 will "fixup" the
         * actual argument pointer passed to the PASE function
         * to point to the address (plus the offset) of the
         * copy of the inputAndResultbuffer that Qp2CallPase2
         * copies to i5/OS PASE storage */
        paseFunctionArglist.inputPasePtr =
        (QP2_ptr64_t)((offsetof(inputAndResultBuffer, inputValue)
            | QP2_ARG_PTR_TOSTACK);

        /* allocate memory from the i5/OS PASE heap for an output
         * argument. Qp2malloc will also set the i5/OS PASE address
         * of the allocated storage in the outputPasePtr
         * argument */
        outputILEPtr = Qp2malloc(10, &(paseFunctionArglist.outputPasePtr));

        /* Call the function in i5/OS PASE */
        rc = callPASE(paseFunction_ptr,
                    &paseFunctionArglist,
                    signature,
                    QP2_RESULT_DWORD,
                    &inputAndResultBuffer,
                    sizeof(inputAndResultBuffer));
        if (rc != 0) {

            printf("output from paseFunction = >%s<\n",
                (char*)outputILEPtr);
            printf("return code from paseFunction = %d\n",
                (int)inputAndResultBuffer.result);
        } /* rc != 0 */
    } /* paseFunction_ptr != NULL */
} /* id != 0 */

/* Close the Qp2dlopen instance, and then call Qp2EndPase
 * to end i5/OS PASE in this job */
Qp2dlclose(id);
Qp2EndPase();
return 0;
}

```

Source code for the i5/OS Procedure paseFunction that is called by the ileMain.c program:

```

/* i5/OS PASE function to be called from ILE
 *

```

```

| * Compile with something like:
| * xlc -q64 -c -o paseFunction.o paseFunction.c
| * ld -b64 -o shr64.o -bnoentry -bexpall -bM:SRE -lc paseFunction.o
| * ar -X64 -r /home/joeuser/libpasefn.a shr64.o
| *
| * The ILE side of this example expects to find libpasefn.a in
| * /home/joeuser/libpasefn.a
| *
| * The compiler options -qalign=natural and -qldbl128 are
| * necessary only when interacting with i5/OS ILE programs
| * to force relative 16-byte alignment of type long double
| * (used inside type ILEpointer)
| */
|
| #include <stdlib.h>
| #include <stdio.h>
| int paseFunction(void * inputPtr, void * outputPtr)
| {
|     /* An output string to return from i5/OS PASE to ILE *
|      * this is the string "output" in EBCDIC          */
|     const char outputValue[] = {0x96, 0xa4, 0xa3, 0x97, 0xa4, 0xa3, 0x00};
|
|     printf("Entered paseFunction The input is >%s<\n",
|           (char*)inputPtr);
|
|     /* copy the output results to the outputPtr argument */
|     memcpy(outputPtr, outputValue, sizeof(outputValue));
|
|     return(52); /* return something more interesting than 0 */
| }

```

## Various functions used in the ILE portion of Example 2

### • The startPASE() function

Before i5/OS PASE can be used in a process, it must be started. This is done automatically by calling an i5/OS PASE application main entry point using the APIs, for example, QP2SHELL, QP2TERM, or Qp2RunPase.

However, because this example is calling an i5/OS PASE function exported from a shared library (not a main entry point), you must manually start i5/OS PASE. Two i5/OS PASE starter utilities are available for this purpose: /usr/lib/start32 (to start the 32-bit version of i5/OS PASE) and /usr/lib/start64 (to start the 64-bit version of i5/OS PASE).

Be aware that each i5/OS process can only have a single instance of i5/OS PASE running. The Qp2ptrsize() API can be used to determine whether i5/OS PASE is already running.

- Qp2ptrsize() will return 0 if i5/OS PASE is not currently active in the process.
- Qp2ptrsize() will return 4 if i5/OS PASE is active in 32-bit mode.
- Qp2ptrsize() will return 8 if i5/OS PASE is active in 64-bit mode.

### • The openlib() and findsym() functions

These functions open the i5/OS shared library and obtain a pointer to the function you want to call using the Qp2dlopen() and Qp2dlsym(). These functions are similar to the dlopen() and dlsym() routines on many platforms.

### • Set up arguments for the Qp2CallPase2 call

Before calling Qp2CallPase2() through the callPASE() function, the main() routine sets up the following variables that define the interface between ILE and the i5/OS PASE function:

- The signature-array variable defines the arguments for the i5/OS PASE function. The elements in the array are typically set using the #define found in the qsysinc/h.qp2user include file.



- | – The `paseFunctionArglist` structure contains the ILE variables that the i5/OS PASE run time will map into the arguments that will be passed to the i5/OS PASE function when the function is called. The members in `paseFunctionArglist` correspond to the signature of the i5/OS PASE function declared in the signature array.
- | – The `inputAndResultBuffer` structure contains the ILE variables that the i5/OS PASE run time will use as a sort of shared buffer between ILE and the i5/OS PASE function when the function is called. The first member of the structure (result in this example) will contain the return value from the i5/OS PASE function. This variable must match the result-type argument provided as the fourth argument in the call to the `Qp2CallPase2` API. Anything after this first element represents storage that will be copied into the i5/OS PASE environment when the function is called.
- | In this example, the `inputValue` element of the `inputAndResultBuffer` structure will contain the by-address argument data that will be pointed at by the first argument for the i5/OS PASE function.
- | – This example uses two different ways of setting pointer arguments for the i5/OS PASE function that is being called.
  - | - The second argument to the function, `paseFunctionArglist.outputPasePtr`, is set by calling the `Qp2malloc()` function. `Qp2malloc()` allocates memory from the i5/OS PASE runtime heap and returns both an ILE pointer and an i5/OS PASE pointer to the allocated storage.
  - | - The first argument, `paseFunctionArglist.inputPasePtr`, is set to the offset of the `inputValue` element of the `inputAndResultBuffer` structure that is connected with the `qp2user.h` #define `QP2_ARG_PTR_TOSTACK` by OR.
    - | This tells the i5/OS PASE run time to modify the actual pointer value provided on the call to the i5/OS PASE function with the address where the `inputAndResultBuffer.inputValue` was copied into i5/OS PASE memory.
- | • **The `callPASE()` function**
  - | This function calls the i5/OS PASE function using the `Qp2CallPase2()` API and the arguments set in the `main()` routine.
- | • **End i5/OS PASE in the process**
  - | After the call to the i5/OS PASE function, the `Qp2dlclose()` API is called to unload the i5/OS PASE shared library and `Qp2EndPase()` is called to end the `start64` program called at the beginning of the example.

## Use i5/OS PASE native methods from Java

You can use i5/OS PASE native methods running in the i5/OS PASE environment from your Java programs.

Support for i5/OS PASE native methods includes full use of the native iSeries Java Native Interface (JNI) from i5/OS PASE native methods and the ability to call i5/OS PASE native methods from the native iSeries JVM.

### Related information

IBM i5/OS PASE native methods for Java

## Work with environment variables

i5/OS PASE environment variables are independent of ILE environment variables. Setting a variable in one environment has no effect on the other environment.

However, you can copy variables from ILE into i5/OS PASE, depending on the method you use to run your i5/OS PASE program.

## Environment variables in an interactive i5/OS PASE session

ILE environment variables are passed to i5/OS PASE only when it is started with `QP2SHELL()` and `QP2TERM()`. Use the Work with Environment Variables (`WRKENVVAR`) command to change, add, or delete environment variables as needed before starting i5/OS PASE.

## Environment variables in a called i5/OS PASE session

When i5/OS PASE is started from a program call (with the Qp2RunPase() API), you have complete control over the environment variables. You can pass environment variables that bear no relationship to the ILE environment from which you called the i5/OS PASE program.

## Copy environment variables to ILE before running a CL command

You can copy i5/OS PASE environment variables to the ILE environment before you run a CL command using an option on the systemCL() runtime function. This is also the default behavior of the i5/OS PASE system utility.

### Related reference

“Run i5/OS PASE programs and procedures” on page 18

You can read this topic for information and examples about starting an i5/OS PASE program in a job, and calling i5/OS PASE procedures from your ILE programs.

### Related information

QP2SHELL() and QP2SHELL2()--Run an i5/OS PASE Shell Program

QP2TERM()--Run an i5/OS PASE Terminal Session

systemCL()--Run a CL Command for i5/OS PASE

i5/OS PASE environment variables

## Call i5/OS programs and procedures from your i5/OS PASE programs

i5/OS PASE provides methods for calling ILE procedures, Java programs, OPM programs, i5/OS APIs, and CL commands that give you integrated access to i5/OS functions.

## General configuration requirements for i5/OS programs and procedures

When you make calls from the i5/OS PASE program environment to the i5/OS environment, you should generally ensure that the i5/OS program is compiled with \*CALLER for the activation group, for the following reasons:

- Only code that runs in the activation group that started i5/OS PASE (called by the Qp2RunPase API) can use ILE APIs, such as Qp2CallPase, to interact with the i5/OS PASE program.
- The ILE runtime might end the entire job (also ending i5/OS PASE) if it needs to destroy an activation group in a multithreaded job (and all jobs created by i5/OS PASE fork are multithread-capable). By using ACTGRP(\*CALLER), you can prevent your job from ending before you want it to end.

You can avoid problems with running in a multithread-capable job by using the systemCL() runtime function to run a CL command (including the CALL command) in a separate job that is not multithread-capable.

### Related tasks

“Customize i5/OS PASE programs to use i5/OS functions” on page 15

If you want your AIX application to take advantage of i5/OS functions that are not directly supported by system-supplied i5/OS PASE shared libraries, you need to perform some additional steps to prepare your application.

## Call ILE procedures

You can follow the instructions in this topic to prepare and call ILE procedures from your i5/OS PASE programs.

When you call ILE procedures from your i5/OS PASE programs, you should first prepare the procedure by enabling it for teraspace, converting text to the appropriate CCSID, and setting up variables and structures.

- **Enable ILE procedures for teraspace**

All ILE modules that you call from i5/OS PASE must be compiled with the teraspace option set to \*YES. If your ILE modules are not compiled in this way, you will receive the MCH4433 error message (Invalid storage model for target program &2) in the job log for your i5/OS PASE application.

- **Convert text to appropriate CCSID**

Text being passed between ILE and i5/OS PASE might need to be converted to the appropriate CCSIDs before being passed. Not doing such conversions causes your character variables to contain undecipherable values.

- **Set up variables and structures**

To make calls to ILE from your i5/OS PASE programs, you need to set up variables and structures. You must ensure that the required header files are copied to your AIX system, and you must set up a signature, a result type, and an argument list variable:

- **Header files:** Your i5/OS PASE program should include the header files `as400_types.h` and `as400_protos.h` to make calls to ILE. The `as400_type.h` header file contains the definition of the types used for i5/OS system-unique interfaces.

- **Signature:** The signature structure contains a description of the sequence and types of arguments passed between i5/OS PASE and ILE. The encoding for the types mandated by the ILE procedure that you are calling can be found in the `as400_types.h` header file. If a signature contains fixed-point arguments shorter than 4 bytes or floating point arguments shorter than 8 bytes, your ILE C code needs to be compiled with the following pragma:

```
#pragma argument(ileProcedureName, nowiden)
```

Without this pragma, standard C linking for ILE requires 1- and 2-byte integer arguments to be widened to 4 bytes and requires 4-byte floating-point arguments to be widened to 8 bytes.

- **Result type:** The result type is straightforward and works much like a return type in C.

- **Argument list:** The argument list must be a structure with the correct sequence of fields with types specified by entries in the signature array. You can use the `size_ILEarglist()` and `build_ILEarglist()` APIs to dynamically build the argument list based on the signature.

To call ILE procedures from your i5/OS PASE programs, make the following API calls in your code:

1. Load the bound program into the ILE activation group that is associated with the procedure that started i5/OS PASE. You use the `_ILELOAD()` API to do this.

This step can be unnecessary if the bound program is already active in the activation group that started i5/OS PASE. In this case, you can proceed to the `_ILESYM` step, using a value of zero for the activation mark parameter to search all symbols in all active bound programs in the current activation group.

2. Find the exported symbol in the activation of the ILE bound program and return a 16-byte tagged pointer to the data or procedure for the symbol. You use the `_ILESYM()` API to do this.
3. Call the ILE procedure to transfer control from your i5/OS PASE program to the ILE procedure. You use the `_ILECALL()` or `_ILECALLX()` API to do this.

#### **Related reference**

“Copy header files” on page 16

You can follow the instructions in this topic to copy header files from your iSeries server to an AIX machine.

#### **Related information**

`size_ILEarglist()`--Compute ILE Argument List Size for i5/OS PASE()

`build_ILEarglist()`--Build an ILE Argument List for i5/OS PASE

`_ILELOADX()`--Load an ILE Bound Program for i5/OS PASE

`_ILESVMX()`--Find an Exported ILE Symbol for i5/OS PASE

`_ILECALLX()`--Call an ILE Procedure for i5/OS PASE

ILE Concepts PDF

## Examples: Call ILE procedures:

The code examples in this topic show i5/OS PASE code making a call to an ILE procedure that is part of a service program and the compiler commands that create the programs.

There are two procedures within the following code examples that show i5/OS PASE code making a call to an ILE procedure that is part of a service program, and the compiler commands that create the programs. Each procedure demonstrates different ways of working with an ILE procedure, but both procedures call the same ILE procedure. The first procedure demonstrates building your data structures for the `_ILECALL` API using i5/OS PASE system-provided methods. The second procedure then builds the argument list manually.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 56.

### Example 1: i5/OS PASE C code

Interspersed in the following example code are comments that explain the code. Make sure to read these comments as you enter or review the example.

```
/* Name: PASEtoILE.c
 *
 * You must use compiler options -qalign=natural and -qldb1128
 * to force relative 16-byte alignment of type long double
 * (used inside type ILEpointer)
 */
#include <stdlib.h>
#include <malloc.h>
#include <sys/types.h>
#include <stdio.h>
#include "as400_types.h"
#include "as400_protos.h"

/*
 * init_pid saves the process id (PID) of the process that
 * extracted the ILEpointer addressed by ILEtarget.
 * init_pid is initialized to a value that is not a
 * valid PID to force initialization on the first
 * reference after the exec() of this program
 *
 * If your code uses pthread interfaces, you can
 * alternatively provide a handler registered using
 * pthread_atfork() to re-initialize ILE procedure
 * pointers in the child process and use a pointer or
 * flag in static storage to force reinitialization
 * after exec()
 */

pid_t init_pid = -1;
ILEpointer*ILEtarget; /* pointer to ILE procedure */

/*
 * ROUND_QUAD finds a 16-byte aligned memory
 * location at or beyond a specified address
 */

#define ROUND_QUAD(x) (((size_t)(x) + 0xf) & ~0xf)

/*
 * do_init loads an ILE service program and extracts an
 * ILEpointer to a procedure that is exported by that
 * service program.
 */
```

```

void do_init()
{
    static char ILEtarget_buf[sizeof(ILEpointer) + 15];
    int actmark;
    int rc;

    /* _ILELOAD() loads the service program */
    actmark = _ILELOAD("SHUPE/ILEPASE", ILELOAD_LIBOBJ);
    if (actmark == -1)
        abort();

    /*
     * xlc does not guarantee 16-byte alignment for
     * static variables of any type, so we find an
     * aligned area in an oversized buffer. _ILESYM()
     * extracts an ILE procedure pointer from the
     * service program activation
     */

    ILEtarget = (ILEpointer*)ROUND_QUAD(ILEtarget_buf);
    rc = _ILESYM(ILEtarget, actmark, "ileProcedure");
    if (rc == -1)
        abort();

    /*
     * Save the current PID in static storage so we
     * can determine when to re-initialize (after fork)
     */
    init_pid = getpid();
}

/*
 * "aggregate" is an example of a structure or union
 * data type that is passed as a by-value argument.
 */
typedef struct {
    char    filler[5];
} aggregate;

/*
 * "result_type" and "signature" define the function
 * result type and the sequence and type of all
 * arguments needed for the ILE procedure identified
 * by ILEtarget
 *
 * NOTE: The fact that this argument list contains
 * fixed-point arguments shorter than 4 bytes or
 * floating-point arguments shorter than 8 bytes
 * implies that the target ILE C procedure is compiled
 * with #pragma argument(ileProcedureName, nowiden)
 *
 * Without this pragma, standard C linkage for ILE
 * requires 1-byte and 2-byte integer arguments to be
 * widened to 4-bytes and requires 4-byte floating-point
 * arguments to be widened to 8-bytes
 */
static result_type_tresult_type = RESULT_INT32;
static arg_type_tsignature[] =
{
    ARG_INT32,
    ARG_MEMPTR,
    ARG_FLOAT64,
    ARG_UINT8,    /* requires #pragma nowiden in ILE code */
    sizeof(aggregate),
    ARG_INT16,
    ARG_END
}

```

```

};

/*
 * wrapper_1 accepts the same arguments and returns
 * the same result as the ILE procedure it calls. This
 * example does not require a customized or declared structure
 * for the ILE argument list. This wrapper uses malloc
 * to obtain storage. If an exception or signal occurs,
 * the storage may not be freed. If your program needs
 * to prevent such a storage leak, a signal handler
 * must be built to handle it, or you can use the methods
 * in wrapper_2.
 */
int wrapper_1(int arg1, void *arg2, double arg3,
              char arg4, aggregate arg5, short arg6)
{
    int result;
    /*
     * xlc does not guarantee 16-byte alignment for
     * automatic (stack) variables of any type, but
     * PASE malloc() always returns 16-byte aligned storage.
     * size_ILEarglist() determines how much storage is
     * needed, based on entries in the signature array
     */
    ILEarglist_base *ILEarglist;
    ILEarglist = (ILEarglist_base*)malloc( size_ILEarglist(signature) );

    /*
     * build_ILEarglist() copies argument values into the ILE
     * argument list buffer, based on entries in the signature
     * array.
     */
    build_ILEarglist(ILEarglist,
                    &arg1,
                    signature);

    /*
     * Use a saved PID value to check if the ILEpointer
     * is set. ILE procedure pointers inherited by the
     * child process of a fork() are not usable because
     * they point to an ILE activation group in the parent
     * process
     */
    if (getpid() != init_pid)
        do_init();

    /*
     * _ILECALL calls the ILE procedure. If an exception or signal
     * occurs, the heap allocation is orphaned (storage leak)
     */
    _ILECALL(ILEtarget,
             ILEarglist,
             signature,
             result_type);
    result = ILEarglist->result.s_int32.r_int32;
    if (result == 1) {
        printf("The results of the simple wrapper is: %s\n", (char *)arg2);
    }
    else if (result == 0) printf("ILE received other than 1 or 2 for version.\n");
    else printf("The db file never opened.\n");
    free(ILEarglist);
    return result;
}

/*
 * ILEarglistSt defines the structure of the ILE argument list.
 * xlc provides 16-byte (relative) alignment of ILEpointer

```

```

* member fields because ILEpointer contains a 128-bit long
* double member. Explicit pad fields are only needed in
* front of structure and union types that do not naturally
* fall on ILE-mandated boundaries
*/
typedef struct {
    ILEarglist_base base;
    int32 arg1;
    /* implicit 12-byte pad provided by compiler */
    ILEpointer arg2;
    float64 arg3;
    uint8 arg4;
    char filler[7]; /* pad to 8-byte alignment */
    aggregate arg5; /* 5-byte aggregate (8-byte align) */
    /* implicit 1-byte pad provided by compiler */
    int16 arg6;
} ILEarglistSt;

/*
* wrapper_2 accepts the same arguments and returns
* the same result as the ILE procedure it calls. This
* method uses a customized or declared structure for the
* ILE argument list to improve execution efficiency and
* avoid heap storage leaks if an exception or signal occurs
*/
int wrapper_2(int arg1, void *arg2, double arg3,
              char arg4, aggregate arg5, short arg6)
{
    /*
    * xlc does not guarantee 16-byte alignment for
    * automatic (stack) variables of any type, so we
    * find an aligned area in an oversized buffer
    */
    char ILEarglist_buf[sizeof(ILEarglistSt) + 15];
    ILEarglistSt *ILEarglist = (ILEarglistSt*)ROUND_QUAD(ILEarglist_buf);
    /*
    * Assignment statements are faster than calling
    * build_ILEarglist()
    */
    ILEarglist->arg1 = arg1;
    ILEarglist->arg2.s.addr = (address64_t)arg2;
    ILEarglist->arg3 = arg3;
    ILEarglist->arg4 = arg4;
    ILEarglist->arg5 = arg5;
    ILEarglist->arg6 = arg6;
    /*
    * Use a saved PID value to check if the ILEpointer
    * is set. ILE procedure pointers inherited by the
    * child process of a fork() are not usable because
    * they point to an ILE activation group in the parent
    * process
    */
    if (getpid() != init_pid)
    do_init();
    /*
    * _ILECALL calls the ILE procedure. The stack may
    * be unwound, but no heap storage is orphaned if
    * an exception or signal occurs
    */
    _ILECALL(ILEtarget,
             &ILEarglist->base,
             signature,
             result_type);
    if (ILEarglist->base.result.s_int32.r_int32 == 1)
        printf("The results of best_wrapper function is: %s\n", arg2);
    else if ( ILEarglist->base.result.s_int32.r_int32 == 0)
        printf("ILE received other than 1 or 2 for version.\n");
}

```

```

else printf("The db file never opened.\n");
return ILEarglist->base.result.s_int32.r_int32;
}
void main () {
    int version,
        result2;
    char dbText[ 25 ];
    double dblNumber = 5.999;
    char justChar = 'a';
    short shrtNumber = 3;
    aggregate agg;
    strcpy( dbText, "none" );

    for (version =1; version <= 2; version
        ++) {if(version==" 1) {
            result2="simple_wrapper(version," dbText, dblNumber, justChar, agg, shrtNumber);
        } else {
            result2="best_wrapper(version," dbText, dblNumber, justChar, agg, shrtNumber);
        }
    }
}

```

## Example 2: ILE C code

You now write the ILE C code for this example on your i5/OS system. You need a source physical file in your library in which to write the code. Again, in the ILE example, comments are interspersed. These comments are critical to understanding the code. You should review them as you enter or review the source.

```

#include <stdio.h>
#include <math.h>
#include <recio.h>
#include <iconv.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>

typedef struct {
    char    filler[5];
} aggregate;

#pragma mapinc("datafile","SHUPE/PASEDATA(*all)","both",,,")
#include "datafile"
#pragma argument(ileProcedure, nowiden) /* not necessary */

/*
 * The arguments and function result for this ILE procedure
 * must be equivalent to the values presented to _ILECALL
 * function in the i5/OS PASE program
 */
int ileProcedure(int    arg1,
                 char   *arg2,
                 double  arg3,
                 char   arg4[2],
                 aggregate arg5,
                 short   arg6)
{
    char    fromcode[33];
    char    tocode[33];
    iconv_t cd;    /* conversion descriptor */
    char    *src;
    char    *tgt;
    size_t  srcLen;
    size_t  tgtLen;
    int     result;

```



```

/*
 * Open a conversion descriptor to convert CCSID 37
 * (EBCDIC) to CCSID 819 (ASCII), that is used for
 * any character data returned to the caller
 */
memset(fromcode, 0, sizeof(fromcode));
strcpy(fromcode, "IBMCCSID000370000000");
memset(tocode, 0, sizeof(tocode));
strcpy(tocode, "IBMCCSID00819");
cd = iconv_open(tocode, fromcode);
if (cd.return_value == -1)
{
    printf("iconv_open failed\n");
    return -1;
}
/*
 * If arg1 equals one, return constant text (converted
 * to ASCII) in the buffer addressed by arg2. For any
 * other arg1 value, open a file and read some text,
 * then return that text (converted to ASCII) in the
 * buffer addressed by arg2
 */
if (arg1 == 1)
{
    src = "Sample 1 output text";
    srcLen = strlen(src) + 1;
    tgt = arg2; /* iconv output to arg2 buffer */
    tgtLen = srcLen;
    iconv(cd, &src, &srcLen, &tgt, &tgtLen);

    result = 1;
}
else
{
    FILE *fp;
    fp = fopen("SHUPE/PASEDATA", "r");
    if (!fp) /* if file open error */
    {
        printf("fopen(\"SHUPE/PASEDATA\", \"r\") failed, "
            "errno = %i\n", errno);
        result = 2;
    }
    else
    {
        char buf[25];
        char *string;
        errno = 0;
        string = fgets(buf, sizeof(buf), fp);
        if (!string)
        {
            printf("fgets() EOF or error, errno = %i\n", errno);
            buf[0] = 0; /* null-terminate empty buffer */
        }
        src = buf;
        srcLen = strlen(buf) + 1;
        tgt = arg2; /* iconv output to arg2 buffer */
        tgtLen = srcLen;
        iconv(cd, &src, &srcLen, &tgt, &tgtLen);

        fclose(fp);
    }
    result = 1;
}
/*
 * Close the conversion descriptor, and return the
 * result value determined above

```

```

    */
    iconv_close(cd);
    return result;
}

```

### Example 3: Compiler commands to create the programs

When you compile your i5/OS PASE program, you must use compiler options `-qalign=natural` and `-qldb1128` to force relative 16-byte alignment of type long double, which is used inside type ILEpointer. This alignment is required by ILE in i5/OS. For option `-bI:`, you should enter the path name in which you saved `as400_libc.exp`:

```

xlc -o PASEtoILE -qldb1128 -qalign=natural
    -bI:/afs/rich.xyz.com/usr1/shupe/PASE/as400_libc.exp
    PASEtoILE.c

```

When you compile your ILE C module and service program, compile them with the `teraspace` option. Otherwise, i5/OS PASE cannot interact with them:

```

CRTCMOD MODULE(MYLIB/MYMODULE)
    SRCFILE(MYLIB/SRCPF)
    TERASPACE(*YES *TSIFC)

CRTSRVPGM SRVPGM(MYLIB/MYSRVPGM)
    MODULE(MYLIB/MOMODULE)

```

Finally, you must compile your DDS and propagate at least one record of data:

```

CRTPF FILE(MYLIB/MYDATAFILE)
    SRCFILE(MYLIB/SRCDDSF)
    SRCMBR(MYMEMBERNAME)

```

### Call i5/OS programs from i5/OS PASE

You can take advantage of existing i5/OS programs (\*PGM objects) when you create your i5/OS PASE applications. In addition, you can use the `systemCL()` function to run the CL CALL command.

Use the `_PGMCALL` runtime function to call an i5/OS program from within your i5/OS PASE program.

This method provides for faster processing than the `systemCL()` runtime function, but it does not perform automatic conversion of character string arguments (unless you specify `PGMCALL_ASCII_STRINGS`), and it does not give you the capability of calling the program in a different job.

#### Related tasks

“Run i5/OS commands from i5/OS PASE” on page 38

You can extend the capabilities of your i5/OS PASE program by running control language (CL) commands that use i5/OS functions.

#### Related information

`_PGMCALL()`--Call an i5/OS Program for i5/OS PASE

### Example: Call i5/OS programs from i5/OS PASE:

You can read the example in this topic to learn about calling programs in an i5/OS PASE program using the `_PGMCALL` runtime function.

The following example shows how you call programs in an i5/OS PASE program using the `_PGMCALL` runtime function.

Interspersed in the following example code are comments that explain the code. Make sure to read these comments as you enter or review the example.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 56.

```
/* This example uses the i5/OS PASE _PGMCALL function to call the i5/OS
API QSZRTVPR. The QSZRTVPR API is used to retrieve information about
i5/OS software product loads. Refer to the QSZRTVPR API documentation
for specific information regarding the input and output parameters needed
to call the API */
```

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include "as400_types.h"
```

```
#include "as400_protos.h"
```

```
int main(int argc, char * argv[])
{
```

```
    /* i5/OS API's (including QSZRTVPR) typically expect character
    parameters to be in EBCDIC. However, character constants in
    i5/OS PASE programs are typically in ASCII. So, declare some
    CCSID 37 (EBCDIC) character parameter constants that will be
    needed to call QSZRTVPR */
```

```
    /* format[] is input parameter 3 to QSZRTVPR and is
    initialized to the text 'PRDR0100' in EBCDIC */
```

```
    const char format[] =
        {0xd7, 0xd9, 0xc4, 0xd9, 0xf0, 0xf1, 0xf0, 0xf0};
```

```
    /* proinfo[] is input parameter 4 to QSZRTVPR and is
    initialized to the text '*OPSYS *CUR 0033*CODE ' in EBCDIC
```

```
    This value indicates we want to check the code load for Option 33
    of the currently installed i5/OS release */
```

```
    const char proinfo[] =
        {0x5c, 0xd6, 0xd7, 0xe2, 0xe8, 0xe2, 0x40, 0x5c, 0xc3,
        0xe4, 0xd9, 0x40, 0x40, 0xf0, 0xf0, 0xf3, 0xf3, 0x5c,
        0xc3, 0xd6, 0xc4, 0xc5, 0x40, 0x40, 0x40, 0x40};
```

```
    /* installed will be compared with the "Load State" field of the
    information returned by QSZRTVPR and is initialized to the text
    '90' in EBCDIC */
```

```
    const char installed[] = {0xf9, 0xf0};
```

```
    /* rcvr is the output parameter 1 from QSZRTVPR */
    char rcvr[108];
```

```
    /* rcvrLen is input parameter 2 to QSZRTVPR */
    int rcvrLen = sizeof(rcvr);
```

```
    /* errcode is input parameter 5 to QSZRTVPR */
    struct {
        int bytes_provided;
        int bytes_available;
        char msgid[7];
    } errcode;
```

```
    /* qszrtvpr_pointer will contain the i5/OS 16-byte tagged system
    pointer to QSZRTVPR */
    ILEpointer qszrtvpr_pointer;
```

```
    /* qszrtvpr_argv6 is the array of argument pointers to QSZRTVPR */
    void *qszrtvpr_argv[6];
```

```
    /* return code from _RSLOBJ2 and _PGMCALL functions */
    int rc;
```

```

/* Set the i5/OS pointer to the QSYS/QSZRTVPR *PGM object */
rc = _RSLOBJ2(&qszrtvpr_pointer,
             RSLOBJ_TS_PGM,
             "QSZRTVPR",
             "QSYS");

/* initialize the QSZRTVPR returned info structure */
memset(rcvr, 0, sizeof(rcvr));

/* initialize the QSZRTVPR error code structure */
memset(&errcode, 0, sizeof(errcode));
errcode.bytes_provided = sizeof(errcode);

/* initialize the array of argument pointers for the QSZRTVPR API */
qszrtvpr_argv[0] = &rcvr;
qszrtvpr_argv[1] = &rcvrlen;
qszrtvpr_argv[2] = &format;
qszrtvpr_argv[3] = &proinfo;
qszrtvpr_argv[4] = &errcode;
qszrtvpr_argv[5] = NULL;

/* Call the i5/OS QSZRTVPR API from i5/OS PASE */
rc = _PGMCALL(&qszrtvpr_pointer,
             (void*)&qszrtvpr_argv,
             0);

/* Check the contents of bytes 63-64 of the returned information.
   If they are not '90' (in EBCDIC), the code load is NOT correctly
   installed */
if (memcmp(&rcvr[63], &installed, 2) != 0)
    printf("i5/OS Option 33 is NOT installed\n");
else
    printf("i5/OS Option 33 IS installed\n");

return(0);
}

```

## Run i5/OS commands from i5/OS PASE

You can extend the capabilities of your i5/OS PASE program by running control language (CL) commands that use i5/OS functions.

Use the `systemCL` runtime function to run an i5/OS command from within an i5/OS PASE program.

When you run i5/OS commands from i5/OS PASE, the `systemCL` runtime function handles ASCII-to-EBCDIC conversion of character string arguments, and lets you call the program in a different job.

### Related tasks

“Call i5/OS programs from i5/OS PASE” on page 36

You can take advantage of existing i5/OS programs (\*PGM objects) when you create your i5/OS PASE applications. In addition, you can use the `systemCL()` function to run the CL CALL command.

### Related information

`systemCL()`--Run a CL Command for i5/OS PASE

### Example: Run i5/OS commands from i5/OS PASE:

You can see the example provided in this topic to learn how to run CL commands in an i5/OS PASE program.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 56.

The following example shows how you call commands in an i5/OS PASE program:

```
/* sampleCL.c
   example to demonstrate use of sampleCL to run a CL command
   Compile with a command similar to the following.
   xlc -o sampleCL -I /whatever/pase -BI:/whatever/pase/as400_libc.exp sampleCL.c
   Example program using QP2SHELL() follows.
   call qp2shell ('sampleCL' 'wrkactjob') */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <as400_types.h> /* PASE header */
#include <as400_protos.h> /* PASE header */

void main(int argc, char* argv[])
{
    int rc;

    if (argc!=2)
    {
        printf("usage: %s \"CL command\"\n", argv[0]);
        exit(1);
    }
    printf("running CL command: \"%s\"\n", argv[1]);

    /* process the CL command */
    rc = systemCL(argv[1], /* use first parameter for CL command */
                  SYSTEMCL_MSG_STDOUT
                  SYSTEMCL_MSG_STDERR ); /* collect messages */

    printf("systemCL returned %d. \n", rc);
    if (rc != 0)
    {
        perror("systemCL");
        exit(rc);
    }
}
```

## How i5/OS PASE programs interact with i5/OS

As you customize your i5/OS PASE programs to use i5/OS functions, you need to consider the ways in which your program will interact with them.

### Related tasks

“Customize i5/OS PASE programs to use i5/OS functions” on page 15

If you want your AIX application to take advantage of i5/OS functions that are not directly supported by system-supplied i5/OS PASE shared libraries, you need to perform some additional steps to prepare your application.

## Communications

i5/OS PASE is generally compatible with AIX and Linux in sockets communications.

i5/OS PASE supports the same syntax as AIX for sockets communications. This cannot match other operating systems, such as Linux, in every detail.

i5/OS PASE sockets support is comparable to the AIX implementation of sockets, but i5/OS PASE uses the i5/OS implementation of sockets (instead of the AIX kernel implementation of sockets), and this forces some minor differences from AIX behavior.

The i5/OS implementation of sockets supports both UNIX 98 and Berkeley Software Distributions (BSD) sockets. In most cases, i5/OS PASE resolves differences in these styles by adopting the behavior of the AIX implementation.

In addition, the user profile for a running application must have the \*IOSYSCFG special authority to specify the level parameter as IPPROTO\_IP and the option\_value parameter as IP\_OPTIONS on socket APIs.

### Related information

Socket programming

Berkeley Software Distributions (BSD) compatibility

UNIX 98 compatibility

## Database

i5/OS PASE supports the DB2 UDB for iSeries Call Level Interface (CLI). DB2 CLI on AIX and i5/OS are not proper subsets of each other, so there are minor differences in a few interfaces, and some APIs in one implementation might not exist in another.

Because of this, you should consider the following points:

- Code can be generated, but not tested, on AIX itself. Instead, you must test your code across platforms within i5/OS PASE.
- You must compile with the i5/OS version of header file `sqlcli.h`. A program compiled using the AIX version of this header file will not run in i5/OS PASE.

i5/OS is an EBCDIC encoded system by default, while AIX is based on ASCII. This difference often requires data conversions between the i5/OS database (DB2 UDB for iSeries) and the i5/OS PASE application.

In the i5/OS PASE implementation of the DB2 CLI, i5/OS PASE system-provided library routines automatically perform data conversions from ASCII to Extended Binary Coded Decimal Interchange Code (EBCDIC) and back for character data. The conversions are made based on the tagged CCSID of the data being accessed and the ASCII CCSID under which the i5/OS PASE program is running. If the database is tagged, or if it is tagged with a CCSID of 65535, no automatic conversion takes place. It is left to the application to understand the encoding format of the data and to do any necessary conversion.

## Work with CCSIDs

When you use the `Qp2RunPase()` API, you must explicitly specify the i5/OS PASE CCSID.

You can control the i5/OS PASE CCSID by setting both of these variables in the ILE before you call API program `QP2TERM`, `QP2SHELL`, or `QP2SHELL2`:

- `PASE_LANG`
- `QIBM_PASE_CCSID`

If the ILE omits either or both of these variables, `QP2TERM`, `QP2SHELL`, and `QP2SHELL2` by default set the i5/OS PASE CCSID and i5/OS PASE environment variable `LANG` with the best i5/OS PASE equivalents of the language and CCSID attributes of your job.

Extensions to `libc.a` give the i5/OS PASE application the ability to change the running CCSID of the application, using the `_SETCCSID()` function.

Another extension gives the i5/OS PASE application the ability to override the DB2 CLI internal conversion without changing the CCSID of the application. The `SQLOverrideCCSID400()` function accepts an integer of the override CCSID as a single parameter.

**Note:** The CCSID override function `SQLOverrideCCSID400()` must be called before any other `SQLx()` API for the override to take effect; otherwise, the request is ignored.

## Use the DB2 UDB for iSeries CLI in i5/OS PASE programs

To use DB2 CLI in your i5/OS PASE programs, you need to copy the `sqlcli.h` header file and the `libdb400.exp` export file to your AIX system before you compile your source. The DB2 CLI library routines are in `libdb400.a` for the i5/OS PASE environment, and are implemented using `pthread` interfaces, providing thread safety. Most i5/OS PASE CLI functions call corresponding ILE CLI functions to perform the required operation.

**Note:** When you use the DB2 CLIs in your i5/OS PASE programs, consider the following points:

- `SQLGetSubString` always returns an EBCDIC string when sub-stringing the CLOB/DBCLOB field. The `SQLGetSubString` is used only for LOB data types.
- `SQLTables`, column 4 of the result set (table type), is always returned as EBCDIC.
- To render graphic-typed data in an i5/OS PASE program, the data must be typed in the program as `wchar`; this causes the database to convert from a graphic and pure double-byte character to Unicode/UCS-2. Otherwise, the database converts between the CCSID of the data and the CCSID of the i5/OS job. The database does not support conversion between EBCDIC graphic and the CCSID (either from the `Qp2RunPase()` API or the `SQLOverrideCCSID400()` API).

### Related reference

“Copy header files” on page 16

You can follow the instructions in this topic to copy header files from your iSeries server to an AIX machine.

“Copy export files” on page 17

You can follow the instructions in this topic to copy the export files from your iSeries server to an AIX directory.

### Related information

`QP2TERM()`--Run an i5/OS PASE Terminal Session

`QP2SHELL()` and `QP2SHELL2()`--Run an i5/OS PASE Shell Program

`_SETCCSID()`--Set i5/OS PASE CCSID

`SQLOverrideCCSID400()`--Override SQL CLI CCSID for i5/OS PASE

SQL call level interface

### Example: Call DB2 UDB for iSeries CLI functions in an i5/OS PASE program:

The example in this topic shows an i5/OS PASE program that accesses DB2 UDB for iSeries using the DB2 UDB for iSeries SQL call level interfaces.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 56.

```
/* i5/OS PASE DB2 UDB for iSeries example program
 *
 * To show an example of an i5/OS PASE program that accesses
 * i5/OS DB2 UDB via SQL CLI
 *
 * Program accesses iSeries Access data base, QIWS/QCUSTCDT, that
 * should exist on all systems
 *
 * Change system name, userid, and password in fun_Connect()
 * procedure to valid parms
 *
 * Compilation invocation:
 *
 * xlc -I./include -bI:./include/libdb400.exp -o paseclidb4 paseclidb4.c
 *
 * FTP in binary, run from QP2TERM() terminal shell
 *
 * Output should show all rows with a STATE column match of MN */
```

```

/* Change Activity: */
/* End Change Activity */

#define SQL_MAX_UID_LENGTH 10
#define SQL_MAX_PWD_LENGTH 10
#define SQL_MAX_STM_LENGTH 255

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "sqlcli.h"

SQLRETURN fun_Connect( void );
SQLRETURN fun_DisConnect( void );
SQLRETURN fun_ReleaseEnvHandle( void );
SQLRETURN fun_ReleaseDbcHandle( void );
SQLRETURN fun_ReleaseStmHandle( void );
SQLRETURN fun_Process( void );
SQLRETURN fun_Process2( void );
void fun_PrintError( SQLHSTMT );

SQLRETURN nml_ReturnCode;
SQLHENV nml_HandleToEnvironment;
SQLHDBC nml_HandleToDatabaseConnection;
SQLHSTMT nml_HandleToSqlStatement;
SQLINTEGER Nmi_vParam;
SQLINTEGER Nmi_RecordNumberToFetch = 0;
SQLCHAR chs_SqlStatement01[ SQL_MAX_STM_LENGTH + 1 ];
SQLINTEGER nmi_PcbValue;
SQLINTEGER nmi_vParam;
char *pStateName = "MN";

void main( ) {
    static
        char*pszId = "main()";
        SQLRETURN nml_ConnectionStatus;
        SQLRETURN nml_ProcessStatus;

        nml_ConnectionStatus = fun_Connect();
        if ( nml_ConnectionStatus == SQL_SUCCESS ) {
            printf( "%s: fun_Connect() succeeded\n", pszId );
        } else {
            printf( "%s: fun_Connect() failed\n", pszId );
            exit( -1 );
        } /* endif */

        printf( "%s: Perform query\n", pszId );
        nml_ProcessStatus = fun_Process();
        printf( "%s: Query complete\n", pszId );
        nml_ConnectionStatus = fun_DisConnect();
        if ( nml_ConnectionStatus == SQL_SUCCESS ) {
            printf( "%s: fun_DisConnect() succeeded\n", pszId );
        } else {
            printf( "%s: fun_DisConnect() failed\n", pszId );
            exit( -1 );
        } /* endif */

        printf( "%s: normal exit\n", pszId );
    } /* end main */

SQLRETURN fun_Connect()
{
    static char *pszId = "fun_Connect()";
    SQLCHAR chs_As400System[ SQL_MAX_DSN_LENGTH ];
    SQLCHAR chs_UserName[ SQL_MAX_UID_LENGTH ];
    SQLCHAR chs_UserPassword[ SQL_MAX_PWD_LENGTH ];
    nml_ReturnCode = SQLAllocEnv( &nml_HandleToEnvironment );

```



```

    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLAllocEnv() succeeded\n", pszId );
        fun_PrintError( SQL_NULL_HSTMT );
        printf( "%s: Terminating\n", pszId );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLAllocEnv() succeeded\n", pszId );
    } /* endif */

    strcpy( chs_As400System, "AS4PASE" );
    strcpy( chs_UserName, "QUSER" );
    strcpy( chs_UserPassword, "QUSER" );
    printf( "%s: Connecting to %s userid %s\n", pszId, chs_As400System, chs_UserName );

    nml_ReturnCode = SQLAllocConnect( nml_HandleToEnvironment,
                                     &nml_HandleToDatabaseConnection );

    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLAllocConnect\n", pszId );
        fun_PrintError( SQL_NULL_HSTMT );
        nml_ReturnCode = fun_ReleaseEnvHandle();
        printf( "%s: Terminating\n", pszId );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLAllocConnect() succeeded\n", pszId );
    } /* endif */

    nml_ReturnCode = SQLConnect( nml_HandleToDatabaseConnection,
                                chs_As400System,
                                SQL_NTS,
                                chs_UserName,
                                SQL_NTS,
                                chs_UserPassword,
                                SQL_NTS );
    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLConnect(%s) failed\n", pszId, chs_As400System );
        fun_PrintError( SQL_NULL_HSTMT );
        nml_ReturnCode = fun_ReleaseDbcHandle();
        nml_ReturnCode = fun_ReleaseEnvHandle();
        printf( "%s: Terminating\n", pszId );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLConnect(%s) succeeded\n", pszId, chs_As400System );
        return SQL_SUCCESS;
    } /* endif */
} /* end fun_Connect */

SQLRETURN fun_Process()
{
    static
        char*pszId = "fun_Process()";
        charcLastName[ 80 ];

    nml_ReturnCode = SQLAllocStmt( nml_HandleToDatabaseConnection,
                                   &nml_HandleToSqlStatement );
    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLAllocStmt() failed\n", pszId );
        fun_PrintError( SQL_NULL_HSTMT );
        printf( "%s: Terminating\n", pszId );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLAllocStmt() succeeded\n", pszId );
    } /* endif */

    strcpy( chs_SqlStatement01, "select LSTNAM, STATE " );
    strcat( chs_SqlStatement01, "from QIWS.QCUSTCDT " );
    strcat( chs_SqlStatement01, "where " );

```

```

strcat( chs_SqlStatement01, "STATE = ? " );

nml_ReturnCode = SQLPrepare( nml_HandleToSqlStatement,
                             chs_SqlStatement01,
                             SQL_NTS );
if ( nml_ReturnCode != SQL_SUCCESS ) {
    printf( "%s: SQLPrepare() failed\n", pszId );
    fun_PrintError( nml_HandleToSqlStatement );
    nml_ReturnCode = fun_ReleaseStmHandle();
    printf( "%s: Terminating\n", pszId );
    return SQL_ERROR;
} else {
    printf( "%s: SQLPrepare() succeeded\n", pszId );
} /* endif */

Nmi_vParam = SQL_TRUE;
nml_ReturnCode = SQLSetStmtOption( nml_HandleToSqlStatement,
                                   SQL_ATTR_CURSOR_SCROLLABLE,
                                   ( SQLINTEGER * ) &Nmi_vParam );
if ( nml_ReturnCode != SQL_SUCCESS ) {
    printf( "%s: SQLSetStmtOption() failed\n", pszId );
    fun_PrintError( nml_HandleToSqlStatement );
    nml_ReturnCode = fun_ReleaseStmHandle();
    printf( "%s: Terminating\n", pszId );
    return SQL_ERROR;
} else {
    printf( "%s: SQLSetStmtOption() succeeded\n", pszId );
} /* endif */

Nmi_vParam = SQL_TRUE;
nml_ReturnCode = SQLSetStmtOption( nml_HandleToSqlStatement,
                                   SQL_ATTR_FOR_FETCH_ONLY,
                                   ( SQLINTEGER * ) &Nmi_vParam );
if ( nml_ReturnCode != SQL_SUCCESS ) {
    printf( "%s: SQLSetStmtOption() failed\n", pszId );
    fun_PrintError( nml_HandleToSqlStatement );
    nml_ReturnCode = fun_ReleaseStmHandle();
    printf( "%s: Terminating\n", pszId );
    return SQL_ERROR;
} else {
    printf( "%s: SQLSetStmtOption() succeeded\n", pszId );
} /* endif */

nmi_PcbValue = 0;
nml_ReturnCode = SQLBindParam( nml_HandleToSqlStatement,
                               1,
                               SQL_CHAR,
                               SQL_CHAR,
                               2,
                               0,
                               ( SQLPOINTER ) pStateName,
                               ( SQLINTEGER * ) &nmi_PcbValue );
if ( nml_ReturnCode != SQL_SUCCESS ) {
    printf( "%s: SQLBindParam() failed\n", pszId );
    fun_PrintError( nml_HandleToSqlStatement );
    nml_ReturnCode = fun_ReleaseStmHandle();
    printf( "%s: Terminating\n", pszId );
    return SQL_ERROR;
} else {
    printf( "%s: SQLBindParam() succeeded\n", pszId );
} /* endif */

nml_ReturnCode = SQLExecute( nml_HandleToSqlStatement );
if ( nml_ReturnCode != SQL_SUCCESS ) {
    printf( "%s: SQLExecute() failed\n", pszId );
    fun_PrintError( nml_HandleToSqlStatement );
    nml_ReturnCode = fun_ReleaseStmHandle();
}

```

```

        printf( "%s: Terminating\n", pszId );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLExecute() succeeded\n", pszId );
    } /* endif */

    nml_ReturnCode = SQLBindCol( nml_HandleToSqlStatement,
                                1,
                                SQL_CHAR,
                                ( SQLPOINTER ) &cLastName,
                                ( SQLINTEGER ) ( 8 ),
                                ( SQLINTEGER * ) &nmi_PcbValue );
    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLBindCol() failed\n", pszId );
        fun_PrintError( nml_HandleToSqlStatement );
        nml_ReturnCode = fun_ReleaseStmHandle();
        printf( "%s: Terminating\n", pszId );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLBindCol() succeeded\n", pszId );
    } /* endif */

    do {
        memset( cLastName, '\0', sizeof( cLastName ) );
        nml_ReturnCode = SQLFetchScroll( nml_HandleToSqlStatement,
                                        SQL_FETCH_NEXT,
                                        Nmi_RecordNumberToFetch );
        if ( nml_ReturnCode == SQL_SUCCESS ) {
            printf( "%s: SQLFetchScroll() succeeded, LastName(%s)\n", pszId, cLastName);
        } else {
            } /*endif */
        } while ( nml_ReturnCode == SQL_SUCCESS );
    if ( nml_ReturnCode != SQL_NO_DATA_FOUND ) {
        printf( "%s: SQLFetchScroll() failed\n", pszId );
        fun_PrintError( nml_HandleToSqlStatement );
        nml_ReturnCode = fun_ReleaseStmHandle();
        printf( "%s: Terminating\n", pszId );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLFetchScroll() completed all rows\n", pszId );
    } /* endif */

    nml_ReturnCode = SQLCloseCursor( nml_HandleToSqlStatement );
    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLCloseCursor() failed\n", pszId );
        fun_PrintError( nml_HandleToSqlStatement );
        nml_ReturnCode = fun_ReleaseStmHandle();
        printf( "%s: Terminating\n", pszId );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLCloseCursor() succeeded\n", pszId );
    } /* endif */

    return SQL_SUCCESS;
} /* end fun_Process */

SQLRETURN fun_DisConnect()
{
    static
        char*pszId = "fun_DisConnect()";

    nml_ReturnCode = SQLDisconnect( nml_HandleToDatabaseConnection );
    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLDisconnect() failed\n", pszId );
        fun_PrintError( SQL_NULL_HSTMT );
        printf( "%s: Terminating\n", pszId );
        return 1;
    }
}

```

```

} else {
    printf( "%s: SQLDisconnect() succeeded\n", pszId );
} /* endif */

    nml_ReturnCode = fun_ReleaseDbcHandle();
    nml_ReturnCode = fun_ReleaseEnvHandle();

    return nml_ReturnCode;
} /* end fun_DisConnect */

SQLRETURN fun_ReleaseEnvHandle()
{
    static
        char*pszId = "fun_ReleaseEnvHandle()";

    nml_ReturnCode = SQLFreeEnv( nml_HandleToEnvironment );
    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLFreeEnv() failed\n", pszId );
        fun_PrintError( SQL_NULL_HSTMT );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLFreeEnv() succeeded\n", pszId );
        return SQL_SUCCESS;
    } /* endif */
} /* end fun_ReleaseEnvHandle */

SQLRETURN fun_ReleaseDbcHandle()
{
    static
        char*pszId = "fun_ReleaseDbcHandle()";

    nml_ReturnCode = SQLFreeConnect( nml_HandleToDatabaseConnection );
    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLFreeConnect() failed\n", pszId );
        fun_PrintError( SQL_NULL_HSTMT );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLFreeConnect() succeeded\n", pszId );
        return SQL_SUCCESS;
    } /* endif */
} /* end fun_ReleaseDbcHandle */

SQLRETURN fun_ReleaseStmHandle()
{
    static
        char*pszId = "fun_ReleaseStmHandle()";

    nml_ReturnCode = SQLFreeStmt( nml_HandleToSqlStatement, SQL_CLOSE );
    if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLFreeStmt() failed\n", pszId );
        fun_PrintError( nml_HandleToSqlStatement );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLFreeStmt() succeeded\n", pszId );
        return SQL_SUCCESS;
    } /* endif */
} /* end fun_ReleaseStmHandle */

void fun_PrintError( SQLHSTMT nml_HandleToSqlStatement )
{
    static
        char*pszId = "fun_PrintError()";

    SQLCHAR chs_SqlState[ SQL_SQLSTATE_SIZE ];
    SQLINTEGER nmi_NativeErrorCode;
    SQLCHAR chs_ErrorMessageText[ SQL_MAX_MESSAGE_LENGTH + 1 ];
    SQLSMALLINT nmi_NumberOfBytes;

```

```

nml_ReturnCode = SQLError( nml_HandleToEnvironment,
                           nml_HandleToDatabaseConnection,
                           nml_HandleToSqlStatement,
                           chs_SqlState,
                           &nmi_NativeErrorCode,
                           chs_ErrorMessageText,
                           sizeof( chs_ErrorMessageText ),
                           &nmi_NumberOfBytes );

if ( nml_ReturnCode != SQL_SUCCESS ) {
    printf( "%s: SQLError() failed\n", pszId );
    return;
} /* endif */

printf( "%s: SqlState - %s\n", pszId, chs_SqlState );
printf( "%s: SqlCode - %d\n", pszId, nmi_NativeErrorCode );
printf( "%s: Error Message:\n", pszId );
printf( "%s: %s\n", pszId, chs_ErrorMessageText );
} /* end fun_PrintError */

```

## Data encoding

Most operating systems, such as AIX and Linux, use ASCII character encoding. Most i5/OS functions use EBCDIC character encoding. You can specify a coded character set identifier (CCSID) value for some i5/OS object types to identify a specific encoding for character data in the object.

i5/OS PASE byte stream files have a CCSID attribute that is used by most system interfaces outside i5/OS PASE to convert text data read from or written to the file as needed. i5/OS PASE does not do CCSID conversion for data read from or written to stream files (consistent with AIX), but it does set the CCSID attribute of any byte stream file created by an i5/OS PASE program to the current i5/OS PASE CCSID value so other system functions can correctly handle ASCII text in the file.

If you use AIX APIs that are shipped in the i5/OS PASE shared libraries, i5/OS PASE handles most of the data conversion for you. i5/OS PASE programs can use `iconv` functions provided in shared library `libiconv.a` for any character data conversions that are not handled automatically by i5/OS PASE runtime. For example, an i5/OS PASE application generally needs to convert character strings to EBCDIC before calling an i5/OS API function (using either `_ILECALLX` or `_PGMCALL`).

### Related concepts

“File systems”

i5/OS PASE programs can access any file or resource that is accessible through the integrated file system, including objects in the QSYS.LIB and QOPT file systems.

## File systems

i5/OS PASE programs can access any file or resource that is accessible through the integrated file system, including objects in the QSYS.LIB and QOPT file systems.

## Buffered input and output

Input and output to and from external devices is buffered on i5/OS; it is handled by input and output processors that deal with blocks of data. Conversely, operating systems, such as AIX and Linux, typically operate with character-by-character (unbuffered) input and output. On i5/OS, only certain input and output signals (for example, the Enter key, function keys, and system request) send an interrupt to the system.

## Data conversion support

i5/OS PASE programs pass ASCII (or UTF-8) path names to the `open()` function to open byte stream files, where the name is automatically converted to the encoding scheme used by i5/OS, but any data read or written from the open file is not converted.

## Use of file descriptors

The i5/OS PASE run time normally uses ILE C run time support for files `stdin`, `stdout`, and `stderr`, which provide consistent behavior for i5/OS PASE and ILE programs.

i5/OS PASE and ILE C use the same streams for standard input and output (`stdin`, `stdout`, and `stderr`). i5/OS PASE programs always access standard input and output using file descriptors 0, 1, and 2. ILE C, however, does not always use integrated file descriptors for `stdin`, `stdout`, and `stderr`, so i5/OS PASE provides a mapping between i5/OS PASE file descriptors and descriptors in the integrated file system. Because of this mapping, i5/OS PASE programs and ILE C programs can use different descriptor numbers to access the same open file.

You can use the i5/OS PASE extension on the `fcntl` function, `F_MAP_XPFFD`, to assign an i5/OS PASE descriptor to an ILE number. This is useful if your i5/OS PASE application needs to do file operations for an ILE descriptor that was not created by i5/OS PASE.

An i5/OS system-unique extension to the `fstatx()` function, `STX_XPFFD_PASE`, allows an i5/OS PASE program to determine the integrated file system descriptor number for an i5/OS PASE file descriptor. Special values (negative numbers) are returned for any i5/OS PASE descriptor attached to ILE C runtime support for files `stdin`, `stdout`, and `stderr`.

If the ILE environment variable `QIBM_USE_DESCRIPTOR_STDIO` is set to Y or I when the `Qp2RunPase()` API is called, i5/OS PASE synchronizes file descriptors 0, 1, and 2 with the integrated file system so that both i5/OS PASE and ILE C programs use the same descriptor numbers for files `stdin`, `stdout`, and `stderr`. When operating in this mode, if either i5/OS PASE code or ILE C code closes or reopens file descriptor 0, 1, or 2, the change affects `stdin`, `stdout`, and `stderr` processing for both environments.

i5/OS PASE run time generally does no character encoding conversion for data read or written through i5/OS PASE file descriptors (including sockets), except that ASCII-to-EBCDIC conversion is done (between the i5/OS PASE CCSID and job default CCSID) for data read from ILE C `stdin` or written to ILE C `stdout` and `stderr`.

Two environment variables control the automatic translation of `stdin`, `stdout`, and `stderr`:

- The variable that generally applies is `QIBM_USE_DESCRIPTOR_STDIO`. When set to Y, the ILE runtime uses file descriptor 0, 1, or 2 for these files.
- The i5/OS PASE system-specific environment variable is `QIBM_PASE_DESCRIPTOR_STDIO`. It has values of B for binary and T for text.

ASCII-to-EBCDIC conversion for i5/OS PASE `stdin`, `stdout`, and `stderr` is disabled if the ILE environment variable `QIBM_USE_DESCRIPTOR_STDIO` is set to Y and `QIBM_PASE_DESCRIPTOR_STDIO` is set to B (allowing binary data to be read from `stdin` and written to `stdout` or `stderr`). The default for `QIBM_PASE_DESCRIPTOR_STDIO` is T for text. This value causes translation of EBCDIC to ASCII.

### Related concepts

“Data encoding” on page 47

Most operating systems, such as AIX and Linux, use ASCII character encoding. Most i5/OS functions use EBCDIC character encoding. You can specify a coded character set identifier (CCSID) value for some i5/OS object types to identify a specific encoding for character data in the object.

### Related information

Integrated file system

## Globalization

Because the i5/OS PASE run time is based on the AIX run time, i5/OS PASE programs can use the same rich set of programming interfaces for locales, character string manipulation, date and time services, message catalogs, and character encoding conversions supported on AIX.

i5/OS PASE supports the interfaces in AIX run time for managing the locale that an application uses and for performing locale-sensitive functions (such as `ctype` and `strcoll`), including support for both single-byte and multibyte character encoding.

i5/OS PASE includes a subset of AIX locales, which provide support for a large number of countries and languages using industry-standard encoding (code sets ISO8859-x), code set IBM-1250, and code set UTF-8. i5/OS PASE provides support for the Euro in three different ways: IBM-1252 locales and ISO 8859-15 locales (both of which use single-byte encodings), and UTF-8 locales.

**Note:** Locale support for i5/OS PASE is independent of either form of locale support used by ILE C programs (object types `*CLD` and `*LOCALE`). In addition to internal structural differences, none of the existing shipped locales for ILE C programs supports ASCII.

## Create new locales

i5/OS PASE does not ship a utility to create new locales. However, you can create locales for use in i5/OS PASE on an AIX system with the `localedef` utility.

## Change locales

When an i5/OS PASE application changes locales, generally it also should change the i5/OS PASE CCSID (using the `_SETCCSID` runtime function) to match the encoding for the new locale. This ensures that any character data interface arguments are correctly interpreted by i5/OS PASE run time (and possibly converted when calling an EBCDIC system service). You can use the `cstoccsid` runtime function to determine what CCSID corresponds to a code set name.

The i5/OS PASE run time sets the CCSID tag on any file created by an i5/OS PASE program to the current i5/OS PASE CCSID value (supplied either when the program is started or using the most recent `_SETCCSID` value).

You should use UTF-8 locales for i5/OS PASE applications that support Japanese, Korean, Traditional Chinese, and Simplified Chinese. i5/OS includes other locales for these languages, but the system does not support setting the i5/OS PASE CCSID to match the encoding for IBM-eucXX code sets. Using UTF-8 support might require converting file data that might be stored in other encoding (such as Shift-JIS) when the application runs on other platforms.

## Where i5/OS PASE conversion objects and locales are stored

Conversion objects and locales for i5/OS PASE are packaged with i5/OS language feature codes. When you install i5/OS PASE, only those locales that are associated with installed i5/OS language features are created.

All i5/OS PASE locales use ASCII or UTF-8 character encoding; therefore, all i5/OS PASE run time works in ASCII (or UTF-8).

### Related tasks

“Install i5/OS PASE” on page 5

You can follow the instructions in this topic to install i5/OS PASE on your server.

### Related information

i5/OS globalization

i5/OS PASE locales

`_SETCCSID()`--Set i5/OS PASE CCSID

## Message services

i5/OS PASE signals and ILE signals are independent, so it is not possible to directly call a handler for one signal type by raising the other type of signal.



You can use the i5/OS PASE Qp2SignalPase() API to post corresponding i5/OS PASE signals for any ILE signal that you receive. The QP2SHELL() program and the i5/OS PASE fork() function always set up handlers to map every ILE signal to a corresponding i5/OS PASE signal.

The system automatically converts any i5/OS exception message sent to the program message queue of a call running the Qp2RunPase, Qp2CallPase, or Qp2CallPase2 API to a corresponding i5/OS PASE signal. An i5/OS PASE application can therefore handle any i5/OS exception by handling the i5/OS PASE signal that the system converts it to.

i5/OS PASE provides the following runtime functions that give you direct control over i5/OS message handling:

- QMHSNDM
- QMHSNDM1
- QMHSNDPM
- QMHSNDPM1
- QMHSNDPM2
- QMHRCVM
- QMHRCVM1
- QMHRCVPM
- QMHRCVPM1
- QMHRCVPM2

See runtime functions for details on these functions.

## i5/OS message support

i5/OS provides message support in a variety of contexts:

- **Job logs.** Your job log contains any messages issued by i5/OS or your application while it is running or being compiled. To look at a job log, type DSPJOBLOG on a command line. When the Display Job Log display screen appears, press F10, followed by Shift + F6. These key combinations result in the Display All Messages display screen being displayed and set to the most recent messages. To view the details of any particular message, move the cursor to the message you want to know more about and press F1.
- **Work with active jobs.** The Work with Active Jobs (WRKACTJOB) command is useful for examining jobs and job stacks on the i5/OS.

### Related information

Qp2SignalPase()--Post an i5/OS PASE Signal

Runtime functions for use by i5/OS PASE programs

Work with active jobs (WRKACTJOB)

Work management

i5/OS PASE signal handling

## Print output from i5/OS PASE applications

You can use the QShell Rfile utility to read and write output from i5/OS PASE shells.

The following example writes the contents of stream file mydoc.ps to spooled printer device file QPRINT as unconverted ASCII data, and then uses the CL LPR command to send the spooled file to another system:

```
before='ovrprtq qprint devtype(*userascii) spool(*yes)'\
after="lpr file(qprint) system(usrchprt01) prtq('rchdps') transform(*no)"
cat -c mydoc.ps | Rfile -wbQ -c "$before" -C "$after" qprint
```



## Related information

Rfile - Read or write record files

## Pseudo-terminal

i5/OS PASE supports both AT&T and Berkeley Software Distributions (BSD) style devices. From a programming perspective, these devices work in i5/OS PASE in the same way that they work on AIX.

i5/OS PASE allows a maximum of 1024 instances for AT&T style devices, and a maximum of 592 BSD style devices. When the system is started, the first 32 instances of each device type are created automatically.

## Configure PTY devices in i5/OS PASE

On AIX, an administrator uses `smitt` to configure the number of available devices of each type. In i5/OS PASE, these devices are configured in the following way:

- For AT&T-style devices, i5/OS PASE supports autoconfiguration. If the first 32 instances are in use and an application tries to open another instance, the CHRSF device is created in the integrated file system automatically, up to the limit of 1024 devices.
- For BSD-style devices, you must create the CHRSF devices manually, using the i5/OS PASE `mknod` utility. To do this, you need to know the major numbers for the BSD subordinate and BSD primary devices as well as the naming convention. The following example shell script shows how to create additional BSD pseudo-terminal (PTY) devices. It creates them in groups of 16.

**Note:** By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 56.

```
#!/Q0penSys/usr/bin/ksh

prefix="pqrstuvwxyzABCDEFGHIJKLMN0PQRSTUVWXYZ"
bsd_tty_major=32949
bsd_pty_major=32948

if [ $# -lt 1 ]
then
    echo "usage: $(basename $0) ptyN "
    exit 10
fi

function mkdev {
    if [ ! -e $1 ]
    then
        mnod $1 c $2 $3
        chown QSYS $1
        chmod 0666 $1
    fi
}

while [ "$1" ]
do
    N=${1##pty}
    if [ "$N" = "$1" -o "$N" = "" -o $N -lt 0 -o $N -gt 36 ]
    then
        echo "skipping: \"$1\": not valid, must be in the form ptyN where: 0 <= N <= 36"
        shift
        continue
    fi

    minor=$((N * 16))
    pre=$(expr "$prefix" : ".\{$N\}\(.\)")

    echo "creating /dev/[pt]ty${pre}0 - /dev/[pt]ty${pre}f"
    for i in 0 1 2 3 4 5 6 7 8 9 a b c d e f
```

```

do
  echo ".\c"
  mkdev /dev/pty${pre}${i} $bsd_pty_major $minor
  echo ".\c"
  mkdev /dev/tty${pre}${i} $bsd_tty_major $minor
  minor=$((minor + 1))
done
echo ""

shift
done

```

For more information about PTY devices, see the AIX documentation Web site.

## Security

From a security point of view, i5/OS PASE programs are subject to the same security restrictions as any other program on i5/OS.

To run an i5/OS PASE program on i5/OS, you must have authority to the AIX binary in the integrated file system. You must also have the proper level of authority to each of the resources that your program accesses, or the program will receive an error when you attempt to access those resources.

The following information is particularly important when you run i5/OS PASE programs.

## User profiles and authority management

System authorization management is based on user profiles that are also objects. All objects created on the system are owned by a specific user. Each operation or access to an object is verified by the system to ensure the user's authority. The owner or appropriately authorized user profiles can delegate various types of authorities to operate on an object to other user profiles. Authority checking is provided uniformly to all types of objects.

The object authorization mechanism provides various levels of control. A user's authority can be limited to exactly what is needed. Files stored in the QOpenSys file system are authorized in the same manner as UNIX files. The following table shows the relationship between UNIX permissions and the security values used on i5/OS database files. On i5/OS, \*OBJOPR is *Use object* authority; \*EXCLUDE is *No authority*. \*READ, \*ADD, \*UPD, \*DLT, and \*EXECUTE are data authorities. You need \*EXECUTE authority (and sometimes \*READ authority) to a file to run it as an i5/OS PASE program.

| UNIX permission | *OBJOPR | *READ | *ADD | *UPD | *DLT | *EXECUTE |
|-----------------|---------|-------|------|------|------|----------|
| r(read)         | X       | X     | -    | -    | -    | -        |
| w(write)        | X       | -     | X    | X    | X    | -        |
| x(execute)      | X       | -     | -    | -    | -    | X        |
| No authority    | -       | -     | -    | -    | -    | -        |

## User profiles in i5/OS PASE

On i5/OS, authentication information is stored in individual *profiles* rather than in such files as `/etc/passwd`. Users and groups have profiles. All of these profiles share one namespace, and each profile must have a unique monospace name. If you pass a lowercase name to the `getpwnam()` or `getgrnam()` API, the system converts the name strings to the expected case.

If you call `getpwuid()` or `getgrgid()` to get the profile name returned, it will be in lowercase, unless you set the i5/OS PASE environment variable `PASE_USRGRP_LOWERCASE=N`, which returns the result in uppercase.

Every user has a user identification (UID). Every group has a group identification (GID). These are defined according to the Portable Operation System Interface X (POSIX) 1003.1 standard. The two numeric spaces are separate, so you can have a user with a UID of 104 and a group with a GID of 104 that are distinct from each other.

i5/OS has a user profile for the security officer, QSECOFR, that has a UID of 0. No other profile can have the UID of 0. QSECOFR is the most privileged profile on the system and, in that sense, acts as the root user. However, i5/OS also provides a set of specific privileges that can be assigned to individual users by system administrators. One of these privileges, \*ALLOBJ, overrides the discretionary access control for file access, for example, which is a typical use of root privileges on operating systems, such as AIX and Linux.

In a ported application that uses root access, it is probably a better security practice to create a specific user profile for the *application user* that can be given \*ALLOBJ authority, therefore avoiding the use of QSECOFR, which has much more privilege than is needed by the single application. Unlike operation systems, such as AIX or Linux, i5/OS does not require group membership for users. The GID of 0 for a user profile on i5/OS means *no group assigned* rather than referring to a group with more privileges.

i5/OS security relies on integrated security built into the system. All accesses to objects must pass a security check. The security check is done with respect to the user profile for which the process runs at the time of the access.

i5/OS PASE relies on giving each process a separate address space to maintain integrity and security. If a resource is not available in your i5/OS PASE address space, you cannot access it. File system security prevents someone from loading a resource into their address space without proper authorization. After it is in the address space, the resource is available to the process regardless of the identity under which the process is running.

An i5/OS PASE program uses system calls to request system functions. System calls for an i5/OS PASE program are handled by i5/OS. This interface gives i5/OS PASE programs only indirect (and safe) access to system internals.

#### **Related information**

Security

### **Work management**

i5/OS handles i5/OS PASE programs in the same way it handles any other job on the system.

#### **Related information**

Work management

---

## **Debug your i5/OS PASE programs**

The i5/OS PASE runtime environment provides library support for the `syslog()` runtime function, and a `syslogd` binary (for more sophisticated message routing). In addition, you can use existing facilities in i5/OS, such as job logs for diagnostic messages and sending severe messages to the i5/OS system operator message queue, QSYSOPR.

Depending on the application, your strategy for debugging an i5/OS PASE application can take different paths:

- If the application does not require any i5/OS integration (for instance, with DB2 UDB for iSeries or with ILE functions), you should first debug the application on AIX.
- Then, you use a combination of i5/OS PASE dbx and i5/OS debug capabilities (such as job logs) to debug the application on i5/OS.

Applications that you have coded to use database or ILE functions cannot be fully tested on AIX, but you can debug the remaining parts of the application on AIX to assure their proper structure and design.

## Use dbx in i5/OS PASE

i5/OS PASE supports the AIX dbx debugger utility. The utility lets you debug related processes, such as parent and child, at the source code level, if they were compiled as such. You can use the Network File System (NFS) to make the AIX source visible to the debugger that runs in i5/OS PASE.

i5/OS PASE support for xterm and aixterm lets you use dbx to debug both the parent and child processes. dbx launches another xterm window with dbx attached to the second process.

For details on dbx, see the AIX documentation Web site. You can also type help on the dbx command line.

## Use i5/OS debugging tools

You can use the following tools on i5/OS to debug your i5/OS PASE applications:

- The iSeries System Debugger provides specific support for i5/OS PASE application debugging.
- The ILE C source debugger is an effective tool for determining problems with your code.

### Related information

iSeries System Debugger

WebSphere Development Studio ILE C/C++ Programmer's Guide PDF

---

## Optimize performance

To achieve the best performance, be sure to store your application binaries in the local stream file system.

It is much slower to start i5/OS PASE programs if your binaries (base program and libraries) are outside of the local stream file system because file mapping cannot be done.

If you run an application in i5/OS PASE that performs a large number of fork() operations, it will not run as fast as it runs on AIX. This is because each i5/OS PASE fork() operation starts a new i5/OS job, which can have a significant impact on performance.

See the Performance topic in the System Management category for information about collecting and analyzing performance data.

---

## Examples

These examples have been provided in the i5/OS PASE information.

**Note:** By using the code examples, you agree to the terms of the "Code license and disclaimer information" on page 56.

### Run i5/OS PASE programs and procedures from ILE programs

- Run an i5/OSPASE program from an ILE program
- Call an i5/OS PASE procedure from an ILE program

### Call i5/OS programs from i5/OS PASE programs

- Call ILE procedures from an i5/OS PASE program
- Call i5/OS programs from i5/OS PASE
- Run CL commands from i5/OS PASE

## Use DB2 UDB for iSeries functions in i5/OS PASE programs


- Call DB2 UDB for iSeries Call Level Interfaces in an i5/OS PASE program

---





## Related information for i5/OS PASE

You can read this topic for the information center topics and Web sites that relate to the i5/OS PASE topic.

### IBM Redbooks™ and Redpapers

Bringing PHP to your iSeries server  The step-by-step implementation discussed in this Redpaper involves the CGI version of the Hypertext Preprocessor (PHP) running in i5/OS Portable Application Solutions Environment (i5/OS PASE).

### Web sites

- Enablement roadmaps & resources   
(<http://www.ibm.com/servers/enable/site/porting/index.html>)  
This Web site compares i5/OS PASE to other solutions for porting your applications to iSeries servers.
- i5/OS PASE   
(<http://www.ibm.com/servers/enable/site/porting/series/pase/index.html>)  
This Web site provides information about porting applications to iSeries servers with i5/OS PASE.
- API Analysis Tool   
(<http://www.ibm.com/servers/enable/site/porting/series/overview/apitool.html>)  
The analysis tool provides detailed information about how your application's use of AIX commands, APIs, and utilities is supported by i5/OS PASE.
- AIX documentation   
(<http://www.ibm.com/servers/aix/library/>)  
This Web site provides information about AIX commands and utilities.

### News groups

The i5/OS PASE news group (<news://news.software.ibm.com/ibm.software.iseries.pase>) discusses user questions and answers relating to i5/OS PASE.

### Other information

- i5/OS PASE APIs  
See this topic for details about the following general categories of i5/OS PASE APIs:
  - Callable program APIs
  - ILE procedure APIs
  - Runtime functions for use by i5/OS PASE programsYou must call a system API to run an i5/OS PASE program. The system provides both callable program APIs and ILE procedure APIs to run i5/OS PASE programs. The callable program APIs can be easier to use, but do not offer all the controls available with the ILE procedure APIs.
- i5/OS PASE shells and utilities  
i5/OS PASE includes three shells (Korn, Bourne, and C Shell) and nearly 200 utilities that run as i5/OS PASE programs. i5/OS PASE shells and utilities provide an extensible scripting environment that includes a large number of industry-standard and de facto-standard commands.
- i5/OS PASE commands

Most i5/OS PASE commands described in this topic support the same options and provide the same behavior as AIX commands. In addition to the i5/OS PASE commands, each i5/OS PASE shell supports a number of built-in commands (such as `cd`, `exec`, and `if`).

- i5/OS PASE runtime libraries


i5/OS PASE run time supports a large subset of the interfaces provided by AIX run time. Most runtime interfaces supported by i5/OS PASE provide the same options and behavior as AIX. The i5/OS PASE runtime libraries are installed (as symbolic links) in `/usr/lib`.

## Saving PDF files

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

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

## Downloading Adobe Reader

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

---

## Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

---

## Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation  
Licensing  
2-31 Roppongi 3-chome, Minato-ku  
Tokyo 106-0032, Japan

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation



Software Interoperability Coordinator, Department YBWA  
3605 Highway 52 N  
Rochester, MN 55901  
U.S.A.

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

- | The licensed program described in this information and all licensed material available for it are provided
- | by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement,
- | IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. \_enter the year or years\_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Programming Interface Information

This i5/OS PASE publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.



---

## Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

- | AIX
- | e(logo)server
- | eServer
- | i5/OS
- | IBM
- | IBM (logo)
- | iSeries
- | pSeries
- | AFS
- | DFS
- | Integrated Language Environment
- | NetServer
- | PartnerWorld
- | POWER
- | PowerPC
- | DB2
- | DB2 Universal Database
- | OS/400

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

---

## Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

**Personal Use:** You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

**Commercial Use:** You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.





Printed in USA