iSeries

OS/400 PASE

IBM

eserver

iSeries

OS/400 PASE

© Copyright International Business Machines Corporation 1998, 2003. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

OS/400 PASE . . . 1
What's new for V5R2 . . . 1
Print this topic . . . 3
Get started with OS/400 PASE . . . 4
  What is OS/400 PASE? . . . 4
  When is OS/400 PASE a useful option for application development? . . . 5
  Install OS/400 PASE . . . 6
Plan for OS/400 PASE . . . 7
Prepare programs to run in OS/400 PASE . . . 8
  Analyze your program's compatibility with OS/400 PASE . . . 9
  Compile your AIX source . . . 10
  Copy the OS/400 PASE program to your iSeries server . . . 11
  Customize OS/400 PASE programs to use OS/400 functions . . . 14
Use OS/400 PASE programs in the OS/400 environment . . . 16
  Run OS/400 PASE programs and procedures . . . 16
  Call OS/400 programs and procedures from your OS/400 PASE programs . . . 21
  How OS/400 PASE programs interact with OS/400 . . . 31
Troubleshoot OS/400 PASE . . . 45
  Debug your OS/400 PASE programs . . . 45
  Optimize performance . . . 46
Examples . . . 46
Related information about OS/400 PASE . . . 46

© Copyright IBM Corp. 1998, 2003

OS/400 PASE

OS/400(R) Portable Application Solutions Environment (OS/400 PASE) allows you to port AIX(R) applications to the iSeries(TM) server with minimal effort. OS/400 PASE provides an integrated run-time environment that allows you to run selected UNIX(R) applications without the complexity of managing a UNIX system. » OS/400 PASE also provides industry-standard and de facto-standard shells and utilities that provide you with a powerful scripting environment. «

To become more familiar with OS/400 PASE, see the following. You can also find information about what's new in this release and how you can print this topic.

Get started with OS/400 PASE
Gives you an overview of the OS/400 Portable Application Solutions Environment, explains how and when OS/400 PASE can be useful for you, and provides instructions for installing OS/400 PASE on your iSeries server.

Plan for OS/400 PASE
Discusses some of the technical requirements you need to consider before you begin using OS/400 PASE.

Prepare programs to run in OS/400 PASE
Provides instructions for creating, compiling, and copying AIX programs that will run effectively in OS/400 PASE.

Use OS/400 PASE programs in the OS/400 environment
Discusses how you can run OS/400 PASE in the OS/400 environment, how you can call OS/400 programs and ILE procedures from within your OS/400 PASE programs, and how OS/400 PASE programs interact with OS/400 functions such as security, messaging, database, communications, work management, printing, and the integrated language environment(R) (ILE).

Troubleshoot OS/400 PASE
Provides information you can use to correct errors in your OS/400 PASE programs and to make them run more efficiently and effectively.

Examples
Provides links to each of the examples in this information and a code example disclaimer that you should read before you use the examples.

Related information about OS/400 PASE
Shows you where you can go in the iSeries Information Center for detailed information about OS/400 PASE APIs, libraries, and utilities. Provides links to additional information outside of the Information Center about OS/400 PASE and AIX.

What's new for V5R2

Here are some of the significant enhancements and changes for OS/400 PASE in V5R2:

- OS/400 PASE supports the installation of either of two AIX compiler products in the OS/400 PASE environment, eliminating the requirement to test and compile your OS/400 PASE programs on AIX. See Compile your AIX source for details.
- Pseudo terminal (PTY) support and UNIX-style job control. See Pseudo terminal (PTY) for details.
- Over 100 new utilities. See the OS/400 PASE Shells and Utilities topic for a complete listing.
- The following new libraries were added. See the OS/400 PASE Run-time Libraries topic for a complete listing.

© Copyright IBM Corp. 1998, 2003

libcur.a	AIX legacy Curses library
libg.a	Debug support
libgair4.a	Internal X Windows support
libl.a	lex support
libld.a	Object File Access Routine library
libm.a	IEEE Math library
libPW.a	Programmers Workbench library
libxcurses.a	Curses library
libXi.a	X Windows input processing
libXtst.a	X Windows testing support
liby.a	yacc support

- A new message (MCH3204) appears in the job log for any unsupported system call used by an OS/400 PASE program. This message text includes the name of the system call and the instruction address that caused the error.
- New and changed OS/400 PASE run-time functions:
  - _CVTERRNO (convert OS/400 PASE errno to ILE errno)
  - _ILECALLX (enhanced ILE procedure call)
  - _PMGCALL (call an OS/400 program)
  - _RETURN (return without exiting OS/400 PASE)
  - _RSLOBJ, _RSLOBJ2 (resolve to an OS/400 object)
  - _STRLEN_SPP, _STRCPY_SPP (string manipulation using 16-byte ILE pointers)
  - Qp2paseCCSID (retrieve OS/400 PASE CCSID)
  - Qp2jobCCSID (retrieve job default CCSID from last time OS/400 PASE CCSID set)
  - faccessx
  - fchdir
  - fclear
  - fclear
  - getaddrinfo, getnameinfo
  - getcontext, setcontext
  - getpri, getpriority, setpriority
  - getprocs64, getthrds64
  - gettimer, settimer
  - msem_init, msem_lock, msleep, msem_unlock, msem_remove
  - pread, pwrite
  - setgroups
  - sigstack, sigaltstack (alternate signal stack)
  - statpriv
  - statvfs, fstatvfs
  - sync
  - ustat
- New and changed (ILE) APIs for OS/400 PASE:
  - QP2SHELL2 (like QP2SHELL, but runs in caller's activation group)
  - Qp2ptrsize (retrieve OS/400 PASE pointer size)
  - Qp2paseCCSID (retrieve OS/400 PASE CCSID)
  - Qp2jobCCSID (retrieve job default CCSID when OS/400 PASE CCSID last set)

- Qp2errnop (locate OS/400 PASE errno for current thread)
- Qp2malloc (allocate OS/400 PASE heap memory)
- Qp2free (free OS/400 PASE heap memory)
- Qp2dlopen (dynamically load an OS/400 PASE module)
- Qp2dlsym (find a symbol in a module opened by OS/400 PASE dlopen)
- Qp2dlclose (close and unload a module loaded by OS/400 PASE dlopen)
- Qp2dlerror (retrieve error information for the last dynamic load operation)
- Qp2CallPase (and Qp2CallPase2) ehancements for by-address arguments and results, and to call an OS/400 PASE procedure in a thread that was not started by OS/400 PASE

- OS/400 PASE locales (and other files for globalization support) are now packaged with OS/400 language feature codes; see Globalization for details. In addition, over 200 new files for X Windows processing of different keyboards and character sets, plus the following 65 new locales. See the OS/400 PASE Locales topic for a complete listing.

```
AR_AE.UTF-8         ES_CO.UTF-8         de_AT.8859-15
AR_BH.UTF-8         ES_MX.UTF-8         de_AT.8859-15@euro
AR_EG.UTF-8         ES_PE.UTF-8         de_LU.8859-15
AR_JO.UTF-8         ES_PR.UTF-8         de_LU.8859-15@euro
AR_KW.UTF-8         ES_UY.UTF-8         en_CA.8859-15
AR_LB.UTF-8         ES_VE.UTF-8         en_IE.8859-15
AR_OM.UTF-8         FR_LU.UTF-8         en_IE.8859-15@euro
AR_QA.UTF-8         FR_LU.UTF-8@euro    en_IN.8859-15
AR_SA.UTF-8         HI_IN.UTF-8         en_NZ.8859-15
AR_SY.UTF-8         SH_YU.UTF-8         es_AR.8859-15
AR_TN.UTF-8         SR_YU.UTF-8         es_CL.8859-15
DE_AT.UTF-8         ar_AE.ISO8859-6     es_CO.8859-15
DE_AT.UTF-8@euro    ar_BH.ISO8859-6     es_MX.8859-15
DE_LU.UTF-8         ar_EG.ISO8859-6     es_PE.8859-15
DE_LU.UTF-8@euro    ar_JO.ISO8859-6     es_PR.8859-15
EN_CA.UTF-8         ar_KW.ISO8859-6     es_UY.8859-15
EN_IE.UTF-8         ar_LB.ISO8859-6     es_VE.8859-15
EN_IE.UTF-8@euro    ar_OM.ISO8859-6     fr_LU.8859-15
EN_IN.UTF-8         ar_QA.ISO8859-6     fr_LU.8859-15@euro
EN_NZ.UTF-8         ar_SA.ISO8859-6     sh_YU.ISO8859-2
ES_AR.UTF-8         ar_SY.ISO8859-6     sr_YU.ISO8859-5
ES_CL.UTF-8         ar_TN.ISO8859-6
```

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 .

Print this topic

To view or download the PDF version, select OS/400 PASE information (about 211 KB or 52 pages).

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 **Save Target As...**
3. Navigate to the directory in which you would like to save the PDF.

4. Click **Save**.

Downloading Adobe Acrobat Reader

If you need Adobe Acrobat Reader to view or print these PDFs, you can download a copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html) .

Get started with OS/400 PASE

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 and AS/400e(TM) servers. As your business moves into an increasingly open computing environment, you are likely finding 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 OS/400 operating systems.

This is where OS/400 Portable Application Solutions Environment (OS/400 PASE) helps. OS/400 PASE lets you run many of your AIX application binaries on OS/400 with little or no change, and effectively expands your platform solution portfolio.

See the following topics to learn more about OS/400 PASE:

- What is OS/400 PASE?
- When is OS/400 PASE a useful option for application development?
- Install OS/400 PASE

What is OS/400 PASE?

OS/400 Portable Application Solutions Environment (OS/400 PASE) is an integrated run-time environment for AIX applications running on OS/400. 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. OS/400 PASE supports the direct execution of PowerPC(TM) machine instructions, so it does not have the drawbacks of an environment that only emulates the machine instructions.

OS/400 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 OS/400 job
- Use OS/400 system functions such as file systems, security, and sockets

Keep in mind that OS/400 PASE is not a UNIX operating system on OS/400. OS/400 PASE is designed to run AIX programs on OS/400 with little or no change. Programs from any other UNIX-based environment need to be written such that they could be compiled on AIX as the first step toward running in OS/400 PASE.

The OS/400 PASE integrated run-time runs on the Licensed Internal Code (LIC) kernel on the iSeries server. The system provides integration of many common OS/400 functions across OS/400 PASE and other run-time environments (including ILE and Java(TM)). OS/400 PASE implements a broad subset of AIX system calls. » System support for OS/400 PASE enforces system security and integrity by controlling what memory an OS/400 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 may run in OS/400 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 OS/400 application integration in your program design (for instance, with CL commands), you can minimize configuration concerns for your application users.

OS/400 PASE adds another porting option for solutions developers who want to share in the success of the OS/400 marketplace. By providing a means to cut porting time significantly, OS/400 PASE can improve the time to market and return on investment for solutions developers.

A broad subset of AIX technology on OS/400

OS/400 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 nonthreadsafe)
- Fortran run-time (both threadsafe and nonthreadsafe)
- `pthreads` threading package
- `iconv` services for data conversion
- Berkeley Software Distributions (BSD) equivalent support
- X-windows 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 OS/400 PASE, and then run on OS/400.

» Alternatively, you can install either of the following products in the OS/400 PASE environment to develop, compile, build, and run your applications completely within OS/400 PASE:

- IBM(R) VisualAge C++ Professional for AIX (Version 6)
- IBM C for AIX (Version 6)

See Compile your AIX source for details. «

» OS/400 PASE also includes the Korn, Bourne, and C shells and nearly 200 utilities that provide a powerful scripting environment. See OS/400 PASE Shells and Utilities for more information. «

OS/400 PASE uses IBM investment in a common processor technology for the AIX and OS/400 operating systems. The PowerPC processor switches from OS/400 mode into AIX mode to run an application in the OS/400 PASE run-time.

Applications running in OS/400 PASE are integrated with the OS/400 Integrated File System and DB2(R) Universal Database(R) for iSeries. They can call (and be called by) Java and Integrated Language Environment (ILE) applications. In general, they can take advantage of all aspects of the OS/400 operations environment, such as security, messaging, communication, and backup and recovery. At the same time, they take advantage of application interfaces that are derived from AIX interfaces.

When is OS/400 PASE a useful option for application development?

OS/400 PASE provides considerable flexibility when you are deciding how to port your AIX applications to the iSeries server. Of course, OS/400 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 OS/400 PASE is an analysis of the application—the APIs, libraries, and utilities that it uses and how effectively it will run on OS/400. The IBM

PartnerWorld(TM) team 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 on how the analysis tool fits into the procedures for porting applications to OS/400 PASE, see Prepare programs to run in OS/400 PASE.

Characteristics of a potential OS/400 PASE application

Here are some useful guidelines that you might consider when making the decision whether or not to use OS/400 PASE:

- **Is the AIX application highly compute-intensive?**
  OS/400 PASE provides a good environment for running computationally intensive applications on iSeries servers by providing highly optimized math libraries.
- **Does the application rely heavily on functions that are supported only in OS/400 PASE (or only partially supported in ILE), such as `fork()`, X-Windows, or pseudo-terminal (PTY) support?**

  OS/400 PASE provides support for `fork()` and `exec()`, which do not currently exist on the OS/400 system (except through `spawn()`, which incorporates the `fork()` function with the `exec()` function).
- **Does the application use a complicated AIX-based build process or testing environment?**
  OS/400 PASE lets you use AIX-based build processes, which are especially useful when you have an existing, complicated process that is not readily transferred onto a new platform.
- **Does the application have dependencies on an ASCII character set?**
  OS/400 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?**
  OS/400 PASE supports both 32-bit and 64-bit AIX addressing models with low performance cost and the ability to convert integers to pointers.

When OS/400 PASE might not be the best solution

OS/400 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 OS/400 PASE on each invocation or calling an OS/400 PASE procedure in an already-active OS/400 PASE program (using the Qp2CallPase API).
- Code that needs to share memory or namespace between an ILE caller and the library code. An OS/400 PASE program does not implicitly share memory or namespace with ILE code that called it. (However, ILE code that is called from OS/400 PASE can share or use OS/400 PASE memory.)

Install OS/400 PASE

OS/400 PASE is available free of charge on all iSeries servers. It is recommended that you install OS/400 PASE; some system software, such as the enhanced domain name server (DNS) server and the ILE C++ compiler, requires OS/400 PASE support.

To install OS/400 PASE on your server:

1. On an OS/400 command line, enter `GO LICPGM`.
2. Select 11. Install licensed program.
3. Select option 33 (5722SS1 - Portable Application Solutions Environment).

Locale installation

The OS/400 PASE product installs only the locale objects that are associated with the language features that you have installed on OS/400. If you need locales that are not included with the language features on your server, you may need to order and install additional OS/400 language features. See OS/400 PASE Globalization and OS/400 PASE locales for more information.

Licensing note for software developers who are porting an application to OS/400 PASE:

OS/400 PASE provides a subset of the AIX run-time libraries on the OS/400 system. The OS/400 license authorizes you to use any library code shipped with OS/400. This license does not imply a license to AIX libraries that were not shipped with OS/400 PASE. All AIX products are separately licensed by IBM.

As you begin porting your own applications to OS/400 PASE, you may find that your application has dependencies on AIX libraries that were not shipped with OS/400 PASE. Before porting these libraries to the OS/400 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 may be necessary to work with IBM or a third party to port additional middleware dependencies to the OS/400 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 sales representative, one of the IBM porting centers, the Custom Technology Center in Rochester, or PartnerWorld for Developers.

Plan for OS/400 PASE

OS/400 PASE provides an AIX run-time environment on OS/400 that lets you port your AIX applications to the iSeries server with minimal effort. In fact, many AIX programs run in OS/400 PASE with no change. This is because OS/400 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(TM) AIX PowerPC processor.

Some points to keep in mind as you begin to work with OS/400 PASE:

- **There is a correlation between the target release of an AIX binary and the release of OS/400 PASE where the binary will run.**

  If you compile your OS/400 PASE applications on AIX, the application binary created on AIX needs to be compatible with the version of OS/400 PASE that you want to the application to run in. The following table shows which AIX binary versions are compatible with different versions of OS/400 PASE. For example, a 32-bit application created for AIX release 4.3 will run on OS/400 PASE V4R5, V5R1, or V5R2, but not on OS/400 PASE V4R4. Similarly, a 64-bit application created for AIX release 4.3 will run on OS/400 PASE V5R1 but not on OS/400 PASE V4R4, V4R5, or V5R2.

AIX release	OS/400 V4R4	OS/400 V4R5	OS/400 V5R1	OS/400 V5R2
4.2 (32-bit)	X	X	X	X
4.3 (32-bit)	-	X	X	X
4.3 (64-bit)	-	-	X	-
5.1 (32- or 64-bit)	-	-	-	X (see note)

  Note: 64-bit Version 5 Release 2 OS/400 PASE applications must be recompiled on AIX 5L(TM) Release 5.1. See Prepare programs to run in OS/400 PASE for details.

- **OS/400 PASE does not provide the AIX kernel on OS/400.**

  Instead, any low-level system functions that are needed by a shared library are routed to the OS/400 kernel or to the integrated OS/400 functions. In this regard, OS/400 PASE bridges the gap across the

AIX and OS/400 platforms: your code uses the same syntax for the APIs in the shared libraries as you would find on AIX, but your OS/400 PASE program runs within an OS/400 job and is managed by OS/400 just like any other OS/400 job.

- **In most cases, the APIs you call in OS/400 PASE behave in exactly the same manner as they do on AIX.**

  Some APIs, however, may behave differently in OS/400 PASE, or may not be supported in OS/400 PASE. Because of this, your plan for preparing OS/400 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 OS/400 PASE.
- **Consider some of the differences that exist between the AIX and OS/400 platforms:**
  - AIX is generally case-sensitive, but certain OS/400 file systems are not.
  - AIX generally uses ASCII for data encoding, but OS/400 generally uses EBCDIC. This will be a consideration if you want to manage the details of calling Integrated Language Environment (ILE) code from your OS/400 PASE program. For example, you must explicitly code OS/400 PASE programs to handle character encoding conversions on strings when you make calls from OS/400 PASE to arbitrary ILE procedures. OS/400 PASE run-time support includes the `iconv_open()`, `iconv()`, and `iconv_close()` functions for character encoding conversion.

    Note: OS/400 PASE and ILE have independent implementations of `iconv()` interfaces, each with its own translation tables. The translations supported by OS/400 PASE `iconv()` support can be modified and extended by users because they are stored as bytestream files stored 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 OS/400 software typically end lines with a carriage return and line feed (CRLF).
  - Some of the scripts and programs you use on AIX may use hard-coded paths to standard utilities, and you will need to modify the path to reflect the paths you will be using in OS/400 PASE. See Analyze your program's compatibility with OS/400 PASE for more information.

OS/400 PASE automatically handles some of these issues. For example, when you use the OS/400 PASE run-time service that the system provides (including any system call or run-time function in a shared library shipped with OS/400 option 33), OS/400 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 (bytestream file or socket).

You can use other low-level functions such as _ILECALL to extend the functionality of your OS/400 PASE program with calls to ILE functions and APIs, but as mentioned above you may need to handle data conversion. Also, coding these extensions into your program will require the use of additional header and export files.

Prepare programs to run in OS/400 PASE

The steps you need to take to prepare AIX programs that run effectively on OS/400 vary with the nature of your program and whether or not you need to use OS/400-unique interfaces and functions.

If you are attempting to port a UNIX application to OS/400 PASE, you must first ensure that the application will compile using an » AIX compiler «. In some cases, you will need to modify your UNIX program to achieve this requirement.

» **Note:** OS/400 PASE V5R2 supports AIX 5L Release 5.1. New 64-bit applications that you want to run in OS/400 PASE must be compiled on AIX 5L Release 5.1, and your existing 64-bit applications must be recompiled. OS/400 PASE supports 32-bit applications from prior releases of AIX without recompiling. «

Step 1: Analyze your program
The first step in this process is recommended in all cases. Use the API Analysis Tool to obtain an in-depth report on the APIs that your program uses, and how you can expect them to perform in OS/400 PASE.

Step 2: Compile your AIX source program
After you have determined your program's suitability as an OS/400 PASE program, and you have made any modifications it might require to run in OS/400 PASE, compile the source. (If your analysis of the AIX program shows that no changes are necessary to run in OS/400 PASE, you do not need to recompile the program.)

» You use an AIX system to compile OS/400 PASE programs, or you can optionally install one of two AIX compiler products in OS/400 PASE to compile your programs in the OS/400 PASE environment. «

Step 3: Copy your program to your iSeries server
» If you compiled your OS/400 PASE program on your AIX system, copy the binary file to your iSeries server. «

Step 4: Optionally, customize your AIX application to use OS/400 interfaces
If you want to customize your AIX application to use OS/400-unique interfaces » and you are compiling your application on AIX «, you must copy one or more OS/400 header or export files to your AIX system before you compile your OS/400 PASE program.

Analyze your program's compatibility with OS/400 PASE

The first step in an assessment of the portability of a UNIX 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 OS/400. It also identifies the interfaces that are standard compliant but supported differently because of the different architecture of OS/400 compared to UNIX 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 UNIX 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 may 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 hard-coded path names:** If you run programs that change credentials or want your programs or scripts to run even when the OS/400 PASE environment variable PASE_EXEC_QOPENSYS=N, you may need to change hard-coded 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 bytestream file, OS/400 PASE automatically searches the /QOpenSys file system for path name `/QOpenSys/usr/bin/ksh`. QShell utility programs are not bytestream files, so OS/400 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`.

Compile your AIX source

When your program uses AIX interfaces only, you compile with any required AIX headers and link with AIX libraries to prepare binaries for OS/400 PASE. Keep in mind that OS/400 PASE does not support applications that are statically bound with AIX system-supplied shared libraries.

OS/400 PASE programs are structurally identical to AIX programs for PowerPC.

OS/400 PASE Option 33 does not include a compiler. You use an AIX system to compile OS/400 PASE programs, or you can optionally install one of two AIX compiler products in OS/400 PASE to compile your programs in the OS/400 PASE environment.

Using AIX compilers on the pSeries server

You can build OS/400 PASE programs using any AIX compiler and linker that generates output that is compatible with the AIX Application Binary Interface (ABI) for PowerPC. OS/400 PASE provides instruction emulation support for binaries that use POWER architecture instructions that do not exist in PowerPC (except for cache-management POWER instructions).

Using AIX compilers in OS/400 PASE

OS/400 PASE supports the installation of either of the following separately-available AIX compilers in the OS/400 PASE environment:

- IBM VisualAge C++ Professional for AIX, Version 6 (5765-F56). (This product includes the IBM C for AIX compiler.)
- IBM C for AIX, Version 6 (5765-F57)

Using these products, you can develop, compile, build, and run your AIX applications entirely within the OS/400 PASE environment on your iSeries server.

For more information about ordering and installing these products, see the VisualAge C++ for AIX compiler installation in OS/400 PASE page on the Application Factory Web site.

Development tools

Many development tools that you use on AIX (for example, ld, ar, make, yacc) are included with OS/400 PASE. See the OS/400 PASE Shells and Utilities topic for details. Many AIX tools from other sources (for instance, the open-source tool `gcc`) can also work in OS/400 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 Application Factory - iSeries Tools for Development 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 MI pointers are 16-byte aligned within structures. Using option -qldbl128 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 OS/400 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 OS/400 PASE programs on an AIX system. If you are using a compiler installed in OS/400 PASE to compile your programs, you would not need to specify compiler options for the locations of OS/400-unique header files or OS/400-unique exports, since these files would be found in their default path locations of /usr/include/ and /usr/lib/ on an OS/400 system.

Example 1: The following command on an AIX system creates an OS/400 PASE program named *testpgm* that can use OS/400-unique interfaces exported by libc.a:

```
xlc -o testpgm -qldbl128 -qlonglong -qalign=natural
        -bI:/mydir/as400_libc.exp testpgm.c
```

This example assumes that the OS/400-unique header files are copied to the AIX directory /usr/include and that the OS/400-unique exports files are copied to AIX directory /mydir.

Example 2: The following example assumes OS/400-unique headers and export files are in /pase/lib:

```
xlc -o as400_test -qldbl128 -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 the compiled application links with threadsafe run-time libraries:

```
xlc_r -o as400_test -qldbl128 -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 OS/400 PASE support for DB2 UDB for iSeries Call Level Interfaces (CLI), you would 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.

Copy the OS/400 PASE program to your iSeries server

Copy AIX binaries that you want to run in OS/400 PASE into the integrated file system. All of the file systems that are available in the integrated file system are available within OS/400 PASE. For more information about the integrated file system, see the Integrated file system topic.

When you move files across platforms, be aware of the following differences that can create problems for you:

- 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.
- AIX and OS/400 use different line terminating characters in text files (for example, in files and shell scripts).

Transferring files

You can use any of the following methods for transferring your OS/400 PASE program and related files to and from your iSeries server:

- File Transfer Protocol (FTP) (see page 12)
- Server Message Block (SMB) (see page 12)
- Remote file systems (see page 12)

Copy programs using File Transfer Protocol (FTP)

You can use the OS/400 FTP daemon and client to transfer a file into or out of the OS/400 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 OS/400 FTP command) when placing files into the integrated file system. This format allows the use of UNIX path names, and transfers the files into stream files. To enter into naming format 1, you can either:

- Change the directory using UNIX 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.

For more information about FTP, see the FTP topic.

Copy programs using Server Message Block (SMB)

OS/400 supports SMB client and server components. With NetServer configured and running, OS/400 PASE has access to SMB servers in the network through the /QNTC file system. On a UNIX platform, a SAMBA server is required to provide the same service. Installing a configured and operational UNIX system, such as AIX, can make directories and files available to OS/400 PASE.

Copy programs using remote file systems

OS/400 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)(TM) and Andrew File System (AFS)(R) (using DFS-to-NFS and AFS-to-NFS translators) so that these file systems can be exported and mounted by OS/400. This, in turn, lets OS/400 PASE applications use these file systems. Security authorization is validated through the OS/400 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.

OS/400 is best used as an NFS server. In this case, you would mount from your AIX system onto a directory in the OS/400 integrated file system, and AIX would write programs directly onto OS/400 when they build.

Note: OS/400 NFS is currently not supported in multithreaded applications.

Case sensitivity

UNIX system interfaces generally differentiate between uppercase and lowercase letters. On OS/400, that is not always the case. You should be aware of several situations in particular where case sensitivity may cause complications with existing code.

Case sensitivity on a directory or file basis depends on the file system you are using on OS/400. The /QOpenSys file system is case-sensitive, and you can create a user-defined file system (UDFS) that is case-sensitive. For information about the characteristics of the various file systems, see the Integrated file system topic (and in particular, the File systems on the integrated file system: comparison topic).

You should also be aware that user IDs and group IDs on OS/400 are always returned in uppercase.

Examples

Following are some examples of problems stemming from case sensitivity that you may encounter.

Example 1: In these examples, 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/QWOBJ*
/qsys.lib/v4r5m0.lib/QWOBJ.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 'QWOBJ*' -print
/qsys.lib/v4r5m0.lib/QWOBJ.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 OS/400 PASE run-time 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
```

«

Line terminating characters in integrated file system files

The AIX applications that are the source for your OS/400 PASE programs expect that lines (for example, in files and shell scripts) will end with a line feed (LF). However, PC software and typical OS/400 software often ends lines with a carriage return and line feed (CRLF).

CRLF used with FTP

One example of where this difference can cause problems is when you use 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. OS/400 FTP always writes exactly what is presented in the data stream and always retains CRLF for text mode, which causes problems with the OS/400 PASE run-time and utilities.

Where possible, use binary mode transfer from a UNIX 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 workaround is offered as a means to strip the CR off of 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(R) Notepad editor). These editors use CRLF as a new line separator, and not the LF that OS/400 PASE expects.

Numerous editors are available (for instance, the ez editor) that do not use CRLF as new line separators. See a listing of various iSeries tools for developers on the IBM PartnerWorld Web site.

Customize OS/400 PASE programs to use OS/400 functions

If you want your AIX application to take advantage of OS/400 functions that are not directly supported by system-supplied OS/400 PASE shared libraries, you need to perform some additional steps to prepare your application.

First, you need to customize your AIX application to call any required OS/400 PASE run-time functions that coordinate your access to the OS/400-unique functions.

Second, if you are compiling your OS/400 PASE programs on an AIX system, you need to perform the following steps before you compile your customized application:

- Copy required OS/400-unique header files to your AIX system
- Copy required OS/400-unique export files to your AIX system

For more information on how you integrate OS/400 PASE with OS/400 functions, see the following topics:

- Call OS/400 programs and functions from your OS/400 PASE programs
- How OS/400 PASE programs interact with OS/400 functions

Copy header files

OS/400 PASE augments standard AIX run-time with header files for OS/400-unique support. These are provided by OS/400 PASE (see page 14) and the OS/400 operating system (see page 15). Copy the header files from your iSeries server to an AIX machine in the header file search path.

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

```
/usr/include
```

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.

See Copy the OS/400 PASE program to your iSeries server for more information about copying files.

Copy OS/400 PASE header files

The OS/400 PASE header files are located in the following OS/400 directory:

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

OS/400 PASE provides the following header files:

as400_protos.h	OS/400 PASE to ILE. Provides miscellaneous OS/400-unique functions.
as400_types.h	Unique OS/400 parameter types for calls to ILE 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. 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 `-qlngdbl128`, `-qalign=natural`, and `-qlonglong` to ensure proper size and alignment of types declared in as400_types.h.
os400msg.h	Functions to send and receive OS/400 messages

Copy OS/400 header files

OS/400-provided header files are located in the following directory:

```
/QIBM/include
```

If your application needs any of the OS/400 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 OS/400 PASE `Rfile` utility.

The following example uses the OS/400 PASE `Rfile` utility to read OS/400 header file `/QIBM/include/qusec.h`, convert the data to the OS/400 PASE CCSID, strip trailing blanks from each line, and then write the result into bytestream file `ascii_qusec.h`:

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

Copy export files

Copy the export files from your iSeries server to an AIX directory.

The export files, located in the following OS/400 directory, are the recommended way to build your applications that require access to OS/400-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.

OS/400 PASE provides the following export files.

as400_libc.exp	Export file for OS/400-unique functions in libc.a The as400_libc.exp file defines all the exports from the OS/400 PASE version of libc.a that are not exported by the AIX versions of those libraries.
libdb400.exp	Export file for OS/400 database functions The libdb400.exp file defines the exports from the OS/400 PASE libdb400.a library (DB2 UDB for iSeries Call Level Interfaces (CLI) support).

See Copy the OS/400 PASE program to your iSeries server for more information about copying files.

OS/400 PASE APIs for accessing OS/400 functions

OS/400 PASE provides a number of APIs for accessing ILE code and other OS/400 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. See OS/400 PASE APIs for details.

Use OS/400 PASE programs in the OS/400 environment

Your OS/400 PASE program can call other OS/400 programs running in your job, and other OS/400 programs can call procedures in your OS/400 PASE program. See the following topics for information about how you incorporate OS/400 PASE programs into your computing environment:

Run OS/400 PASE programs from your OS/400 programs
Provides information and examples about starting an OS/400 PASE program in a job, and calling OS/PASE procedures from your ILE programs.

Call OS/400 programs and functions from your OS/400 PASE programs
Provides information and examples for calling ILE procedures, OS/400 programs, and CL commands from your OS/400 PASE programs

How OS/400 PASE programs interact with OS/400 functions
Provides information about how OS/400 PASE programs use and interact with OS/400 functions.

Run OS/400 PASE programs and procedures

OS/400 PASE provides several methods for running your OS/400 PASE programs.

» **QP2SHELL() and QP2SHELL2()** «
OS/400 programs that run an OS/400 PASE program in the job in which they are called.

QP2TERM()
An OS/400 program that runs OS/400 PASE programs in an interactive shell environment.

Qp2RunPase()
An ILE procedure that you call from within ILE procedures to start and run an OS/400 PASE program.

Qp2CallPase()
An ILE procedure that you call from within ILE procedures to run an OS/400 PASE program in a job where the OS/400 PASE environment is already running.

Work with environment variables explains how the OS/400 PASE environment interacts with the OS/400 environment.

ILE procedures that let you work with OS/400 PASE programs

» OS/400 PASE provides a number of ILE procedure APIs that allow your ILE code to access OS/400 PASE services (without special programming in your OS/400 PASE program):

- Qp2ptrsize
- Qp2jobCCSID
- Qp2paseCCSID
- Qp2errnop
- Qp2malloc
- Qp2free
- Qp2dlopen
- Qp2dlsym
- Qp2dlclose
- Qp2dlerror

See OS/400 PASE ILE Procedure APIs for more information.

Attach to ILE threads

You can call a procedure in an OS/400 PASE program from ILE code that runs in a thread that was not created by OS/400 PASE (for example, a Java thread or a thread created by ILE pthread_create). Qp2CallPase automatically attaches the ILE thread to OS/400 PASE (creating corresponding OS/400 PASE pthread structures), but only if the OS/400 PASE environment variable PASE_THREAD_ATTACH was set to Y when the OS/400 PASE program started.

Return results from OS/400 PASE to OS/400 programs

Using the OS/400 _RETURN() function, you can call an OS/400 PASE program and return results without ending the OS/400 PASE environment. This allows you to start an OS/400 PASE program and then call procedures in that program (using Qp2CallPase) after the QP2SHELL2 (but not QP2SHELL) or Qp2RunPase API returns. «

Run an OS/400 PASE program with QP2SHELL()

You use the Run an OS/400 PASE Shell Program (QP2SHELL or » QP2SHELL2 «) programs to run an OS/400 PASE program from any OS/400 command line and within any high-level language program, batch job, or interactive job. These programs run an OS/400 PASE program in the job that calls it. The name of the OS/400 PASE program is passed as a parameter on the program. See the QP2SHELL() and QP2SHELL2() description for details about how to use this program.

The QP2SHELL() program runs the OS/400 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 OS/400 command line:

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

Passing values into QP2SHELL() using variables

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

```
PGM

DCL  VAR(&CMD)    TYPE(*CHAR)   LEN(20)    VALUE('/QOpenSys/bin/ls')
DCL  VAR(&PARM1)  TYPE(*CHAR)   LEN(10)    VALUE('/')
DCL  VAR(&NULL)   TYPE(*CHAR)   LEN(1)     VALUE(X'00')

                   CHGVAR VAR(&CMD) VALUE(&CMD *TCAT &NULL)
                   CHGVAR VAR(&PARM1) VALUE(&PARM1 *TCAT &NULL)

                   CALL PGM(QP2SHELL) PARM(&CMD &PARM1)

ENDIT:
ENDPGM
```

Run an OS/400 PASE program with QP2TERM()

Start an OS/400 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 OS/400 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 OS/400 PASE program that you want to run, as well as any argument strings to pass to the program.

You can run any OS/400 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.

Run an OS/400 PASE program from within OS/400 programs

Use the Qp2RunPase() API to run an OS/400 PASE program. You specify the program name, argument strings, and environment variables. See the Qp2RunPase() API description for details about how you use it in your ILE programs.

The Qp2RunPase() API runs an OS/400 PASE program in the job where it is invoked. It loads an OS/400 PASE program (including any necessary shared libraries) and then transfers control to the program.

This API gives you more control over how OS/400 PASE runs than QP2SHELL() and QP2TERM().

See the example programs for an example of how you might use this API in your ILE programs.

Examples: Run an OS/400 PASE program from within OS/400 programs: The following ILE program (see disclaimer) calls an OS/400 PASE program. Following this example is an example of the OS/400 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 OS/400
  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
     OS/400 PASE program */
  char *PASE_parm = "My Parm";
  /* Argument list for OS/400 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 OS/400 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 OS/400 PASE */
        arg_list,       /* Arguments for OS/400 PASE program */
        NULL);          /* Environment variable list, not used in this sample */
  }
```

The following OS/400 PASE program (see disclaimer) is called by the above ILE program.

```
#include <stdio.h>

/*******************************************
  Sample:
  A simple OS/400 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 OS/400.
  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 OS/400 PASE program %s. Parameter value is \"%s\".\n", argv[0], argv[1]);

     return 0;
}
```

Call an OS/400 PASE procedure from within OS/400 programs

The Qp2RunPase() API initially starts and runs an OS/400 PASE program in a job. It returns an error if OS/400 PASE is already active in that job.

» To call OS/400 PASE procedures in a job that is already running an OS/400 PASE program, you use the Qp2CallPase() and Qp2CallPase2() APIs.

See Example: Call an OS/400 PASE procedure from within OS/400 programs for an example of how to use the Qp2CallPase() API. «

Example: Call an OS/400 PASE procedure from within OS/400 programs: The following ILE program (see disclaimer) calls an OS/400 PASE procedure.

```
#include <stdio.h>
#include <qp2shell2.h>
#include <qp2user.h>
#define JOB_CCSID 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 OS/400 PASE program
     * /usr/lib/start32, which starts OS/400 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 OS/400 PASE getpid
     * subroutine (exported by shared library libc.a)
     */
    id = Qp2dlopen(NULL, QP2_RTLD_NOW, JOB_CCSID);
    getpid_pase = Qp2dlsym(id, "getpid", JOB_CCSID, NULL);

    /*
     * Call Qp2CallPase to run the OS/400 PASE getpid
     * function, and print the result.  Use Qp2errnop
     * to find and print the OS/400 PASE errno if the
     * function result was -1
     */
    int rc = Qp2CallPase(getpid_pase,
                         NULL,        // no argument list
                         signature,
                         QP2_RESULT_WORD,
                         &result)
    printf("OS/400 PASE getpid() = %i\n", result);
    if (result == -1)
        printf("OS/400 errno = %i\n", *Qp2errnop());

    /*
     *  Close the Qp2dlopen instance, and then call
     *  Qp2EndPase to end OS/400 PASE in this job
     */
    Qp2dlclose(id);
    Qp2EndPase();
    return 0;
}
```

Work with environment variables

OS/400 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 OS/400 PASE, depending on the method you use to run your OS/400 PASE program.

Environment variables in an interactive OS/400 PASE session

ILE environment variables are passed to OS/400 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 OS/400 PASE.

Environment variables in a called OS/400 PASE session

When OS/400 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 OS/400 PASE program.

» Copy environment variables to ILE before running a CL command

You can copy OS/400 PASE environment variables to the ILE environment before you run a CL command using an option on the systemCL run-time function. This is also the default behavior of the OS/400 PASE `system` utility. «

See the OS/400 PASE Environment Variables topic for additional information.

Call OS/400 programs and procedures from your OS/400 PASE programs

OS/400 PASE provides methods for calling ILE procedures, Java programs, OPM programs, OS/400 APIs, and CL commands that give you integrated access to OS/400 functions.

The following topics provide instructions and examples for making calls from the OS/400 PASE environment.

Call ILE procedures
Before you can call ILE procedures from OS/400 PASE, you need to make sure that your ILE procedures are set up to handle calls from OS/400 PASE programs. You also need to set up the program variables and structures in your compiled AIX program.

» **Call OS/400 programs**
You can call OS/400 programs from within your OS/400 PASE program. «

Run OS/400 commands
You can run CL commands from within your OS/400 PASE program.

General configuration requirements for OS/400 programs and procedures

» When you make calls from the OS/400 PASE program environment to the OS/400 environment, you should generally ensure that the OS/400 program is compiled with *CALLER for the activation group, for the following reasons:

- Only code that runs in the activation group that started OS/400 PASE (called by the Qp2RunPase API) can use ILE APIs such as Qp2CallPase to interact with the OS/400 PASE program.
- The ILE run-time may end the entire job (also ending OS/400 PASE) if it needs to destroy an activation group in a multithreaded job (and all jobs created by OS/400 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` run-time function to run a CL command (including the CALL command) in a separate job that is not multithread-capable.

Call ILE procedures

To call ILE procedures from your OS/400 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 OS/400 PASE. You use the _ILELOAD() API to do this. » This step may be unnecessary if the bound program is already active in the activation group that started OS/400 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 OS/400 PASE program to the ILE procedure. You use the _ILECALL() or _ILECALLX() API to do this.

In addition, you must perform the following tasks when you call ILE procedures from your OS/400 PASE programs:

- Enable ILE procedures for teraspace
- Convert text to appropriate CCSID
- Set up variables and structures

Enable ILE procedures for teraspace

All ILE modules that you call from OS/400 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 OS/400 PASE application. See the ILE Concepts book for more information.

Convert text to appropriate CCSID

Text being passed between ILE and OS/400 PASE may need to be converted to the appropriate CCSIDs before being passed. Not doing such conversions will cause your character variables to contain undecipherable values.

Set up variables and structures

To make calls to ILE from your OS/400 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 OS/400 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 OS/400-unique interfaces.
- **Signature:** The signature structure contains a description of the sequence and types of arguments passed between OS/400 PASE and ILE. The encoding for the types mandated by the ILE procedure 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 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.
- **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.

See Examples: Calling ILE procedures for examples that illustrate the process for making calls to ILE procedures from OS/400 PASE.

Examples: Calling ILE procedures: The following code examples (see disclaimer) show OS/400 PASE code (see page 22) making a call to an ILE procedure (see page 26) that is part of a service program, and the compiler commands that create the programs (see page 28). Within the example, there are two UNIX procedures. 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 OS/400 PASE-provided methods. The second procedure then builds the argument list manually.

OS/400 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 -qldbl128
 * 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
 * ILE pointer 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 ILE pointer
     * 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 ILE pointer
     * 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);
        }
    }
}
```

ILE C code

You now write the ILE C code for this example on your OS/400 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 OS/400 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;
}
```

Compiler commands to create the programs

When you compile your OS/400 PASE program, you must use compiler options -qalign=natural and -qldbl128 to force relative 16-byte alignment of type long double, which is used inside type ILEpointer. This alignment is required by ILE in OS/400. For option -bI:, you should enter the path name in which you saved as400_libc.exp:

```
xlc -o PASEtoILE -qldbl128 -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, OS/400 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 OS/400 programs from OS/400 PASE

You can take advantage of existing OS/400 programs (*PGM objects) when you create your OS/400 PASE applications:

- Use the systemCL function to run the CL CALL command. See Run OS/400 commands from OS/400 PASE for information and an example.
- Use the _PGMCALL run-time function to call an OS/400 program from within your OS/400 PASE program. This method provides for faster execution than the systemCL run-time function, but it does not perform automatic conversion of character string arguments, and it does not give you the capability of calling the program in a different job.

See Example: Calling OS/400 programs from OS/400 PASE for an example of how you call commands in an OS/400 PASE program using the _PGMCALL run-time function. «

Example: Call OS/400 programs from OS/400 PASE: The following example (see disclaimer) shows how you call programs in an OS/400 PASE program using the _PGMCALL run-time function:

```
/* This example uses the OS/400 PASE _PGMCALL function to call
   the OS/400 API QSZRTVPR.

   The QSZRTVPR API is used to retrieve information about OS/400
   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[])
{

   /* OS/400 API's (including QSZRTVPR) typically expect character
      parameters to be in EBCDIC.  However, character constants in
      OS/400 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};

   /* prodinfo[] 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 OS/400 release */
   const char prodinfo[] =
      {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, 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 OS/400 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 OS/400 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] = &prodinfo;
   qszrtvpr_argv[4] = &errcode;
   qszrtvpr_argv[5] = NULL;

   /* Call the OS/400 QSZRTVPR API from OS/400 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("OS/400 Option 33 is NOT installed\n");
   else
      printf("OS/400 Option 33 IS installed\n");

   return(0);
}
```

Run OS/400 commands from OS/400 PASE

You can extend the capabilities of your OS/400 PASE program by running control language (CL) commands that use OS/400 functions. You use the systemCL run-time function to run an OS/400 command from within an OS/400 PASE program.

When you run OS/400 commands from OS/400 PASE, the systemCL run-time function automatically handles ASCII-to-EBCDIC conversion of character string arguments, and lets you call the program in a different job.

See Example: Run OS/400 commands from OS/400 PASE) for an example of how you run CL commands in an OS/400 PASE program.

Example: Run OS/400 commands from OS/400 PASE: The following example (see disclaimer) shows how you call commands in an OS/400 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 OS/400 PASE programs interact with OS/400

As you customize your OS/400 PASE programs to use OS/400 functions, you need to consider the ways in which your program will interact with them. The following topics provide basic guidance, and provide links to detailed OS/400 system information in the iSeries Information Center:

- Communications
- Database
- Data encoding

- File systems
- Globalization
- Message services
- Printing
- Pseudo-terminal (PTY)
- Security
- Work management

Communications

OS/400 PASE supports the same syntax as AIX for sockets communications. This may not match other UNIX(TM) systems in every detail.

OS/400 PASE sockets support is comparable to the AIX implementation of sockets, but OS/400 PASE uses the OS/400 implementation of sockets (instead of the AIX kernel implementation of sockets), and this forces some minor differences from AIX behavior.

The OS/400 implementation of sockets supports both UNIX 98 and Berkeley Software Distributions (BSD) sockets. In most cases, OS/400 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. See the Socket programming topic for details about the use of sockets on OS/400. In particular, see the Berkeley Software Distributions (BSD) compatibility and UNIX 98 compatibility topics.

Database

OS/400 PASE supports the DB2 UDB for iSeries Call Level Interfaces (CLI). DB2 CLI on AIX and OS/400 are not proper subsets of each other, so there are minor differences in a few interfaces, and some APIs in one implementation may 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 OS/400 PASE.
- You must compile with the OS/400 version of header file `sqlcli.h`. A program compiled using the AIX version of this header file will not run in OS/400 PASE.

OS/400 is an EBCDIC encoded system by default, while AIX is based on ASCII. This difference often requires data conversions between the OS/400 database (DB2 UDB for iSeries) and the OS/400 PASE application.

In the OS/400 PASE implementation of the DB2 CLI, OS/400 PASE-provided library routines automatically perform data conversions from ASCII to 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 OS/400 PASE program is running. If the database is tagged or 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.

Working with CCSIDs

When you use the Qp2RunPase() API, you must explicitly specify the OS/400 PASE CCSID.

You can control the OS/400 PASE CCSID by setting both of these variables in the ILE environment before you call API program QP2TERM, QP2SHELL, or QP2SHELL2:

- PASE_LANG
- QIBM_PASE_CCSID

If the ILE environment omits either or both of these variables, QP2TERM, QP2SHELL, and QP2SHELL2 by default set the OS/400 PASE CCSID and OS/400 PASE environment variable LANG with the best OS/400 PASE equivalents of the language and CCSID attributes of your job.

See the QP2TERM() and QP2SHELL() program descriptions for more information.

Extensions to libc.a give the OS/400 PASE application the ability to change the running CCSID of the application, using the _SETCCSID() function.

Another extension gives the OS/400 PASE application the ability to override the DB2 CLI internal conversion without changing the CCSID of the application. The SQLOverrideCCSID400() function accepts as a single parameter an integer of the override CCSID.

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

Using the DB2 UDB for iSeries CLI in OS/400 PASE programs

To use DB2 CLI in your OS/400 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 OS/400 PASE environment, and are implemented using pthread interfaces, providing threadsafety. Most OS/400 PASE CLI functions call corresponding ILE CLI functions to perform the desired operation.

See the DB2 UDB for iSeries SQL Call Level Interface (ODBC) topic for more information on the DB2 UDB call level interfaces.

See Example: Call DB2 UDB for iSeries CLI functions in an OS/400 PASE program for an example of how OS/400 PASE accesses DB2 UDB for iSeries using the DB2 UDB for iSeries SQL call level interfaces.

Example: Call DB2 UDB for iSeries CLI functions in an OS/400 PASE program: The following example (see disclaimer) shows an OS/400 PASE program that accesses DB2 UDB for iSeries using the DB2 UDB for iSeries SQL call level interfaces.

```
/* OS/400 PASE DB2 UDB for iSeries example program
 *
 * To show an example of an OS/400 PASE program that accesses
 * OS/400 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 UNIX systems use ASCII character encoding. » Most OS/400 functions use EBCDIC character encoding. You can specify a CCSID (Coded Character Set Identifier) value for some OS/400 object types to identify a specific encoding for character data in the object. «

» OS/400 PASE bytestream files have a CCSID attribute that is used by most system interfaces outside OS/400 PASE to convert text data read from or written to the file as needed. OS/400 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 bytestream file created by an OS/400 PASE program to the current OS/400 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 OS/400 PASE shared libraries, OS/400 PASE handles most of the data conversion for you. » OS/400 PASE programs can use `iconv` functions provided in shared library `libiconv.a` for any character data conversions that are not handled automatically by OS/400 PASE run-time. For example, an OS/400 PASE application generally needs to convert character strings to EBCDIC before calling an OS/400 API function (using either _ILECALLX or _PGMCALL). «

File systems

OS/400 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 OS/400; it is handled by input and output processors that deal with blocks of data. Conversely, UNIX systems typically operate with character-by-character (unbuffered) input and output. On OS/400, 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

OS/400 PASE programs pass ASCII (or UTF-8) path names to the `open` function to open bytestream files, where the name is automatically converted to the encoding scheme used by OS/400, but any data read or written from the open file is not converted.

See Data encoding for more information about data conversion.

Use of file descriptors

The OS/400 PASE run-time normally uses ILE C run-time support for files `stdin`, `stdout`, and `stderr`, which provide consistent behavior for OS/400 PASE and ILE programs.

OS/400 PASE and ILE C use the same streams for standard input and output (stdin, stdout, and stderr). OS/400 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 OS/400 PASE provides a mapping between OS/400 PASE file descriptors and descriptors in the integrated file system. Because of this mapping, OS/400 PASE programs and ILE C programs may use different descriptor numbers to access the same open file.

You can use the OS/400 PASE extension on the `fcntl` function, F_MAP_XPFFD, to assign an OS/400 PASE descriptor to an ILE number. This is useful if your OS/400 PASE application needs to do file operations for an ILE descriptor that was not created by OS/400 PASE.

An OS/400-unique extension to the `fstatx` function, STX_XPFFD_PASE, allows an OS/400 PASE program to determine the integrated file system descriptor number for an OS/400 PASE file descriptor. Special values (negative numbers) are returned for any OS/400 PASE descriptor attached to ILE C run-time 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, OS/400 PASE synchronizes file descriptors 0, 1, and 2 with the integrated file system so that both OS/400 PASE and ILE C programs use the same descriptor numbers for files `stdin`, `stdout`, and `stderr`. When operating in this mode, if either OS/400 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.

OS/400 PASE run-time generally does no character encoding conversion for data read or written through OS/400 PASE file descriptors (including sockets), except that ASCII-to-EBCDIC conversion is done (between the OS/400 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 run-time uses file descriptor 0, 1, or 2 for these files.
- The PASE-specific environment variable is QIBM_PASE_DESCRIPTOR_STDIO. It has values of B for binary and T for text.

ASCII-to-EBCDIC conversion for OS/400 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.

For more information about file systems, see the Integrated file system topic.

Globalization

Because the OS/400 PASE run-time is based on the AIX run-time, OS/400 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.

OS/400 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.

OS/400 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. OS/400 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 OS/400 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.

Creating new locales

OS/400 PASE does not ship a utility to create new locales. However, you can create locales for use in OS/400 PASE on an AIX system with the `localedef` utility.

Changing locales

When an OS/400 PASE application changes locales, it generally should also change the OS/400 PASE CCSID (using the _SETCCSID run-time function) to match the encoding for the new locale. This ensures that any character data interface arguments are correctly interpreted by OS/400 PASE run-time (and possibly converted when calling an EBCDIC system service). You can use the `cstoccsid` run-time function to determine what CCSID corresponds to a code set name.

The OS/400 PASE run-time sets the CCSID tag on any file created by an OS/400 PASE program to the current OS/400 PASE CCSID value (supplied either when the program is started or using the most recent _SETCCSID value).

You should use UTF-8 locales for OS/400 PASE applications that support Japanese, Korean, Traditional Chinese, and Simplified Chinese. OS/400 includes other locales for these languages, but the system does not support setting the OS/400 PASE CCSID to match the encoding for IBM-euc*XX* code sets. Using UTF-8 support may require converting file data that may be stored in other encoding (such as Shift-JIS) when the application runs on other platforms.

Where OS/400 PASE conversion objects and locales are stored

» Conversion objects and locales for OS/400 PASE are packaged with OS/400 language feature codes. When you install OS/400 PASE, only those locales that are associated with installed OS/400 language features are created. «

All OS/400 PASE locales use ASCII or UTF-8 character encoding; therefore, all OS/400 PASE run-time works in ASCII (or UTF-8).

See the Globalization topic for more information about globalization on OS/400.

Message services

OS/400 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 OS/400 PASE Qp2SignalPase() API to post corresponding OS/400 PASE signals for any ILE signal that you receive. The QP2SHELL() program and the OS/400 PASE `fork()` function always set up handlers to map every ILE signal to a corresponding OS/400 PASE signal.

» The system automatically converts any OS/400 exception message sent to the program message queue of an invocation running the Qp2RunPase, Qp2CallPase, or Qp2CallPase2 API to a corresponding OS/400 PASE signal. An OS/400 PASE application can therefore handle any OS/400 exception by handling the OS/400 PASE signal that the system converts it to. «

» OS/400 PASE provides the following runtime functions that give you direct control over OS/400 message handling:

- QMHSNDM
- QMHSNDM1
- QMHSNDPM

- QMHSNDPM1
- QMHSNDPM2
- QMHRCVM
- QMHRCVM1
- QMHRCVPM
- QMHRCVPM1
- QMHRCVPM2

See Run-time functions for details on these functions.

OS/400 message support

OS/400 provides message support in a variety of contexts:

Job logs
Your job log will contain any messages issued by OS/400 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 screen appears, press the F10 key, followed by Shift + F6. These key combinations result in the Display All Messages 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 the F1 key.

Work with active jobs
The Work with Active Jobs (WRKACTJOB) command is useful for examining jobs and job stacks on the OS/400.

For more information about message support on OS/400, see the Work Management topic.

For more information about OS/400 PASE and OS/400 messages, see OS/400 PASE Signal Handling.

Print output from OS/400 PASE applications

You can use the QShell Rfile utility to read and write output from OS/400 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 spool file to another system:

```
before='ovrprtf 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
```

Pseudo-terminal (PTY)

OS/400 PASE supports both AT&T and Berkeley Software Distributions (BSD) style devices. From a programming perspective, these devices work in OS/400 PASE in the same way that they work on AIX.

OS/400 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.

Configuring PTY devices in OS/400 PASE

On AIX, an administrator would use `smit` to configure the number of available devices of each type. In OS/400 PASE, these devices are configured in the following way:

- For AT&T style devices, OS/400 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 OS/400 PASE `mknod` utility. To do this, you will need to know the major numbers for the BSD slave and BSD master devices as well as the naming convention. The following example shell script shows how to create additional BSD PTY devices. It creates them in groups of 16.

```
#!/QOpenSys/usr/bin/ksh

prefix="pqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
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
    mknod $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 pseudo-terminal devices, see the AIX documentation web site.

Security

From a security point of view, OS/400 PASE programs are subject to the same security restrictions as any other program on OS/400. To run an OS/400 PASE program on OS/400, 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 OS/400 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 may delegate to other user profiles various types of authorities to operate on an object. Authority checking is provided uniformly to all types of objects.

The object authorization mechanism provides various levels of control. A user's authority may 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 OS/400 database files. On OS/400, *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 OS/400 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 OS/400 PASE

On OS/400, 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 monocase name. If you pass a lowercase name to the `getpwnam()` or `getgrnam()` APIs, 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 OS/400 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 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.

OS/400 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, OS/400 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 UNIX systems.

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 UNIX systems, OS/400 does not require group membership for users. The GID of 0 for a user profile on OS/400 means *no group assigned* rather than referring to a group with more privileges.

OS/400 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.

OS/400 PASE relies on giving each process a separate address space to maintain integrity and security. If a resource is not available in your OS/400 PASE address space, you cannot access it. File system security prevents someone from loading a resource into their address space without proper authorization. Once in the address space, the resource is available to the process regardless of the identity under which the process is running.

An OS/400 PASE program uses system calls to request system functions. System calls for an OS/400 PASE program are handled by OS/400. This interface gives OS/400 PASE programs only indirect (and safe) access to system internals.

To learn more about security on iSeries servers, see the Security topic.

Work management

OS/400 handles OS/400 PASE programs in the same way it handles any other job on the system. For information about how OS/400 handles jobs, see the Work Management topic.

Troubleshoot OS/400 PASE

This information provides guidance on how you can debug your OS/400 PASE programs and make them run more efficiently and effectively:

- Debug your OS/400 PASE programs
- Optimize performance

Debug your OS/400 PASE programs

The OS/400 PASE run-time environment provides library support for the `syslog()` run-time function, and a `syslogd` binary (for more sophisticated message routing). In addition, you can use existing facilities in OS/400, such as job logs for diagnostic messages and sending severe messages to the OS/400 system operator message queue, QSYSOPR.

Depending on the application, your strategy for debugging an OS/400 PASE application can take different paths:

- If the application does not require any OS/400 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 OS/400 PASE `dbx` and OS/400 debug capabilities (such as job logs) to debug the application on OS/400.

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.

Using dbx in OS/400 PASE

OS/400 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 OS/400 PASE.

» OS/400 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.

Using OS/400 debugging tools

The ILE C source debugger is an effective tool for determining problems with your code. To learn about this tool, see the WebSphere(TM) Development Studio ILE C/C++ Programmer's Guide .

You can also use the iSeries System Debugger to debug your OS/400 PASE code.

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 OS/400 PASE programs if your binaries (base program and libraries) are outside of the local stream file system since file mapping cannot be done.

If you run an application in OS/400 PASE that performs a large number of `fork()` operations, it will not run as fast as it runs on AIX. This is because each OS/400 PASE `fork()` operation starts a new OS/400 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

The following examples have been provided in the OS/400 PASE information. Before you use these examples, read the code example disclaimer below.

Run OS/400 PASE programs and procedures from ILE programs

- Run an OS/400 PASE program from an ILE program
- Call an OS/400 PASE procedure from an ILE program

Call OS/400 programs from OS/400 PASE programs

- Call ILE procedures from an OS/400 PASE program
- Call OS/400 programs from OS/400 PASE
- Run CL commands from OS/400 PASE

Use DB2 UDB for iSeries functions in OS/400 PASE programs

- Call DB2 UDB for iSeries Call Level Interfaces in an OS/400 PASE program

Code example disclaimer

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.

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you "AS IS" without any warranties of any kind. The implied warranties of non-infringement, merchantability and fitness for a particular purpose are expressly disclaimed.

Related information about OS/400 PASE

See the following resources for more information about how you can use OS/400 PASE.

Other iSeries Information Center topics

OS/400 PASE APIs
See this topic for details about the following general categories of OS/400 PASE APIs:

- Callable program APIs
- ILE procedure APIs
- Run-time functions for use by OS/400 PASE programs

You must call a system API to run an OS/400 PASE program. The system provides both callable program APIs and ILE procedure APIs to run OS/400 PASE programs. The callable program APIs can be easier to use, but do not offer all the controls available with the ILE procedure APIs.

OS/400 PASE shells and utilities
OS/400 PASE includes three shells (Korn, Bourne, and C Shell) and nearly 200 utilities that run as OS/400 PASE programs. OS/400 PASE shells and utilities provide an extensible scripting environment that includes a large number of industry-standard and de facto-standard commands.

OS/400 PASE commands
Most OS/400 PASE commands described in this topic support the same options and provide the same behavior as AIX commands. In addition to the OS/400 PASE commands, each OS/400 PASE shell supports a number of built-in commands (such as `cd`, `exec`, and `if`).

OS/400 PASE libraries
OS/400 PASE run-time supports a large subset of the interfaces provided by AIX run-time. Most run-time interfaces supported by OS/400 PASE provide the same options and behavior as AIX. The OS/400 PASE run-time libraries are installed (as symbolic links) in `/usr/lib`.

Web sites

AIX commands, APIs, and utilities that OS/400 PASE supports**:** The API Analysis Tool is the most efficient and effective way to find detailed information about how your application's use of AIX functions is supported by OS/400 PASE.

The Application Factory and OS/400 PASE pages in the IBM PartnerWorld for Developers Web site provide general information about porting applications to iSeries servers with OS/400 PASE. The Application Factory compares OS/400 PASE to other solutions for porting your applications to iSeries servers.

For more information about AIX commands and utilities, see the AIX documentation web site.

Printed in U.S.A.