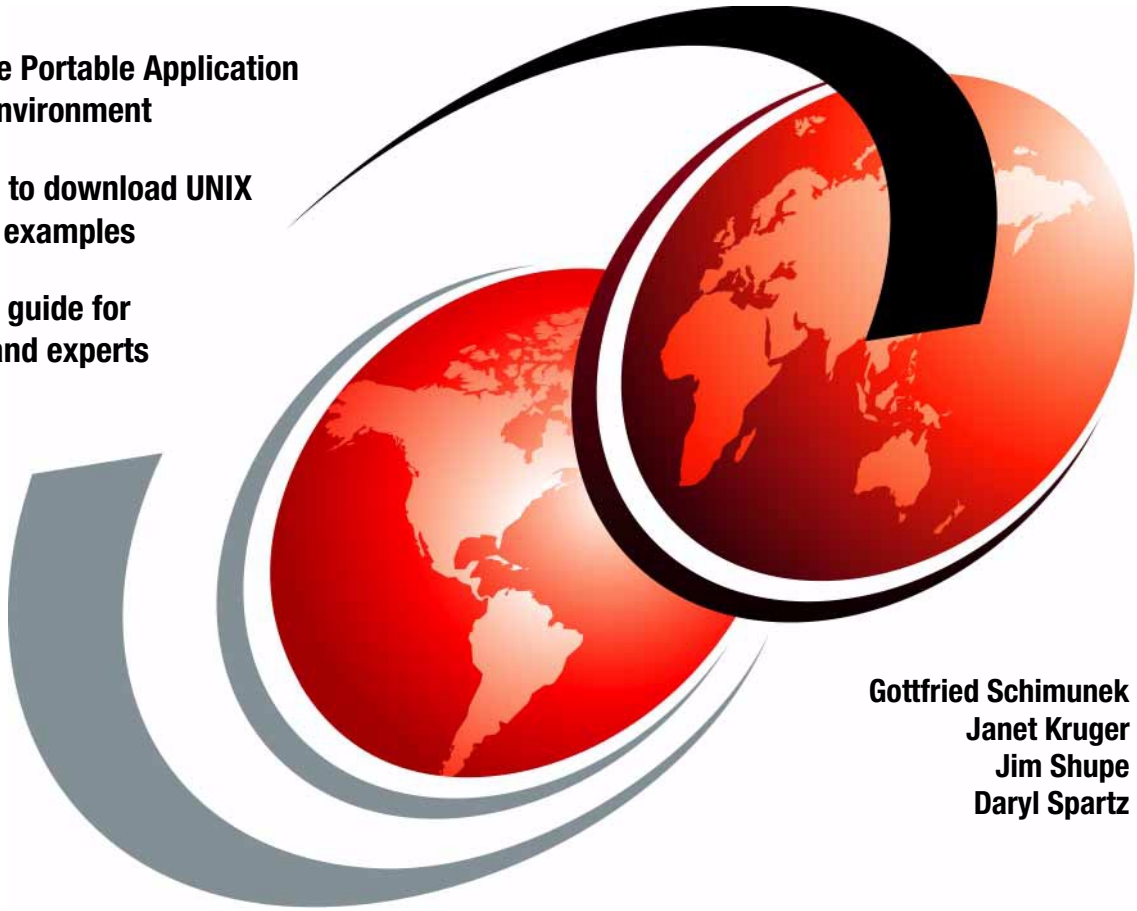
IBM

Porting UNIX Applications

Using AS/400 PASE

- Explores the Portable Application Solutions Environment
- Offers links to download UNIX application examples
- The porting guide for beginners and experts



Gottfried Schimunek
Janet Kruger
Jim Shupe
Daryl Spartz

Redbooks

ibm.com/redbooks

SG24-5970-00

International Technical Support Organization

Porting UNIX Applications Using AS/400 PASE

July 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special notices” on page 217.

First Edition (July 2000)

This edition applies to Version 4 Release 5 of the AS/400 Operating System OS/400, Program Number 5769-SS1 option 33.

Note

This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

Contents

Preface ix
The team that wrote this redbook ix
Comments welcome xi

Part 1. The AS/400 PASE runtime environment 1

Chapter 1. Introduction to the AS/400 PASE runtime environment 3
1.1 Executive overview 3
1.1.1 Positioning of AS/400 PASE 4
1.1.2 Planning a port to AS/400 PASE 6
1.2 Technical overview 7
1.2.1 Hardware considerations 10
1.2.2 Development environment 10
1.2.3 Performance 10

Chapter 2. AS/400 PASE capabilities and limitations 11
2.1 Advantages of AS/400 PASE porting 11
2.1.1 Leveraging existing skills 11
2.1.2 Reducing time to market 11
2.2 Successfully ported applications 12
2.3 Capabilities 14
2.3.1 File systems 14
2.3.2 Shared libraries 14
2.3.3 Runtime library members not supported 16
2.3.4 Runtime symbols 19
2.3.5 Shells and utilities 19
2.3.6 Logging facilities 22
2.3.7 OS/400 access 22
2.4 Limitations 23
2.4.1 Devices 23
2.4.2 File systems 23
2.4.3 Printing 24
2.4.4 Shells 24
2.4.5 Libraries 24
2.4.6 API restrictions 24
2.4.7 Unsupported system calls 31

Chapter 3. Getting started with AS/400 PASE 33
3.1 Requirements 33
3.2 Licensing issues 33
3.3 Recommendations 34

3.4 General AS/400 PASE instructions 34
3.4.1 General system setup 35
3.4.2 Checking system support 35
3.4.3 Preparing the application 35
3.4.4 Testing the application 35
3.4.5 Creating your AS/400 solution 36

Part 2. AS/400 PASE from a UNIX perspective 37

Chapter 4. AS/400 architecture from a UNIX perspective 39
4.1 Object-oriented architecture 39
4.2 Addressing and storage management 40
4.2.1 Job and process structure 41
4.3 Library and address resolution 42
4.4 User profiles and authority management 42
4.5 Character sets and terminal I/O 43
4.5.1 EBCDIC versus ASCII 43
4.5.2 Buffered versus unbuffered I/O 44
4.5.3 I/O controllers versus device drivers 44
4.6 Architectural summary 44

Chapter 5. Application API analysis 45
5.1 Introduction 45
5.2 What you need to do 45
5.2.1 API analysis example 46

Chapter 6. Porting mechanism 57
6.1 Beginning a port 57
6.2 Compiling applications on AIX for AS/400 PASE 58
6.3 Accessing files from AS/400 PASE 60
6.3.1 Using FTP 60
6.3.2 Using SMB 62
6.3.3 Remote file systems 62
6.4 Configuration tips 62
6.5 Starting an AS/400 PASE application 65
6.5.1 Invocation from a 5250 terminal screen: QP2SHELL 65
6.5.2 Invocation from the AS/400 PASE terminal: QP2TERM 69
6.5.3 Invocation from an ILE application: Qp2RunPase 71
6.5.4 Invocation from an ILE application: Qp2CallPase 79
6.5.5 Calling a procedure: Qp2SignalPase 82
6.6 Debugging an AS/400 PASE application 84
6.6.1 Using dbx in AS/400 PASE 84
6.6.2 AS/400 PASE unsupported system calls 85

Chapter 7. Database porting with AS/400 PASE 87
7.1 Data encoding considerations 87
7.2 Known problems. 89
7.3 DB2CLI example program 89

Chapter 8. ILE integration with the AS/400 PASE environment 99
8.1 Shared addressing between AS/400 PASE and single level store . . 100
8.1.1 System structure 101
8.1.2 Memory model and program model 102
8.1.3 File system and socket support 104
8.1.4 Runtime support 105
8.1.5 Development environment 107
8.1.6 Performance. 107
8.2 Calling Java from AS/400 PASE. 108
8.3 Doing callouts to ILE from AS/400 PASE 108
8.3.1 Setting up variables and structures. 109
8.3.2 A two part sample for calling ILE from AS/400 PASE 111
8.3.3 size_ILEarglist() function 120
8.3.4 build_ILEarglist() function 121
8.3.5 _ILELOAD() function 125
8.3.6 _ILESYM() function 127
8.3.7 _ILECALL() function 129
8.3.8 _MEMCPY_WT() and _MEMCPY_WT2() functions 134
8.3.9 _SETSPP() function 135
8.3.10 _CVTSPP() function 136
8.3.11 _SETCCSID() function 137
8.3.12 systemCL function 138
8.4 Calling AS/400 PASE from ILE. 141
8.4.1 Qp2RunPase information 141
8.4.2 Qp2CallPase information 141

Part 3. AS/400 PASE from an AS/400 perspective 143

Chapter 9. UNIX architecture from an AS/400 perspective 145

Chapter 10. AS/400 PASE porting examples. 147
10.1 GNU zip 147
10.2 OpenDX 150
10.3 GNU perl 159

Chapter 11. Work management 169
11.1 Viewing AS/400 PASE programs running on the AS/400 system . . 169

Chapter 12. Problem determination and messages 173
12.1 Available problem determination tools 173
12.1.1 Tools on AIX 173
12.1.2 Tools on OS/400 173
12.2 Where to find messages on the AIX system 179
12.3 Where to find messages on the AS/400 system 179
12.4 Debugging the AS/400 to ILE sample 179

Chapter 13. Security considerations 185
13.1 User profiles in AS/400 PASE 185
13.2 AIX exploitation points 186

Chapter 14. AS/400 PASE globalization 187
14.1 Locale support 187
14.2 File system and sockets support 188
14.3 stdin, stdout, and stderr 189
14.4 Interactive terminal support 189
14.5 Character encoding conversion support 190
14.6 Date and time services 191
14.7 Database support 191
14.8 X-Windows support 192
14.9 Device support 192
14.10 Shells and utilities 192
14.11 AS/400 PASE locales 193
14.12 AS/400 PASE codesets 198

Appendix A. Programming resources 203

Appendix B. The Application Factory 205
B.1 Overview 205
B.2 Around the circle 207
B.2.1 Licensing 207
B.2.2 Installation and packaging 207
B.2.3 Operations 208
B.2.4 AS/400 workloads and performance 209
B.2.5 Security 209
B.2.6 National language and internationalization 209
B.2.7 Problem management 210
B.2.8 Database access 210
B.2.9 Printing 210

Appendix C. AS/400 PASE compared to ILE 211

Appendix D. Special notices 217

Appendix E. Related publications 221
E.1 IBM Redbooks 221
E.2 IBM Redbooks collections. 221
E.3 Other resources 222
E.4 Referenced Web sites. 222

How to get IBM Redbooks 225
IBM Redbooks fax order form 226

Glossary 227

Index 229

IBM Redbooks review 235

Preface

This redbook positions the new AS/400 Portable Application Solutions Environment (PASE), referred to as AS/400 PASE, with the other options available to AS/400 application developers. It helps you determine whether an AIX-based application or module can be easily ported to PASE, or whether you should consider an ILE port.

There are five primary audiences for this redbook:

- Executives and planners who are looking at using AS/400 PASE for portions of their solutions
- UNIX application providers who are considering a port of their application to the AS/400 platform
- Businesses that already leverage both AS/400 and AIX solutions and want to consolidate their solutions into one platform
- AS/400 shops looking to purchase or use an AS/400 PASE application
- AS/400 shops looking to purchase a UNIX solution who want to encourage the vendor of that solution to consider AS/400 delivery and support

With these audiences in mind, we have developed the bulk of this book in three parts:

- Part 1, “The AS/400 PASE runtime environment”, applies to all audiences and provides an overview of the product.
- Part 2, “AS/400 PASE from a UNIX perspective”, targets UNIX application providers and businesses that are looking to port their own in-house applications from AIX to the AS/400 system.
- Part 3, “AS/400 PASE from an AS/400 perspective”, is intended for all technical audiences, including AS/400 system operators.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Rochester Center.

Gottfried Schimunek is a certified Consulting I/T Architect at IBM PartnerWorld for developers in Rochester, Minnesota. He helps IBM Business Partners and Independent Software Providers (ISVs) to understand the newest technologies available on the AS/400 system and provides guidance to implement them into

their applications. He worked for three years in the International Technical Support Organization as an international asignee and authored and published many redbooks on application development and performance. Before joining the ITSO in 1997, Gottfried worked in the Technical Support Center in Germany as a Consultant for IBM Business Partners and Customers as well as for the IBM sales force.

Janet Krueger is a Consulting Software Engineer at D.H. Andrews Group in Rochester, Minnesota. She has 24 years of experience in software development. She holds a Masters of Computer Science from the University of Iowa. Janet worked for IBM in Rochester for 23 years. Her most recent IBM accomplishments were in the AS/400 Partners in Development group, where she created the AS/400 Tools Network program. Janet was the lead architect for PC Support/400 from 1987 through 1993, and acted as a technical liaison between the Novell corporation and the Rochester Programing Lab for three years. Janet joined IBM in 1976, working in programming language development for the System/34. Later, Janet transferred to the programming support area, designing and coding both MVS and VM application development tools for internal IBM use. Janet has been one of the primary strategists on transforming the AS/400 application base into a set of competitive e-business solutions. In 1998, Janet received an AS/400 Division Award for her long-term contributions to AS/400 application modernization. Janet is an award winning COMMON speaker and received COMMON's Distinguished Service Award in 1999.

Daryl Spartz is an Advisory Software Engineer in the Custom Technology Center in Rochester, MN USA. He has 18 years of experience in the system support, mainframe, and PC development fields. He holds Bachelor of Science and Master of Science degrees in Computer Science from Moorhead State University and the University of Minnesota, respectively. He has worked at IBM for 18 years. His areas of expertise include TCP/IP, Internet security with firewalls, C, C++, and a growing interest in Linux.

Jim Shupe is a Staff Software Engineer in the Rochester AS/400 Support Center in Rochester, MN USA. He has eight years of experience in the software development and support field. He holds Bachelor degrees in Computer Science, Scientific and Technical Communication, and a Master degree in Rhetoric and Technical Communication from Michigan Technological University. His areas of expertise include natural language processing, C, C++, Java, and technical writing.

Thanks to the following people for their invaluable contributions to this project:

Patrick Barrett
Tracy Bashore
Pamela Bowen
Tony Cairns
Sam Ellis
Kevin Erickson
Mike Good
Rich Griswold
Jeff Holecek
David Jones
David Larson
Wade Ouren
Charlie Quigg
Dan Sundt
Kay Tate
George Timms
Neil Willis
IBM Rochester

Tarlochan Bimbra
Keane, Rochester

Shigeru Shimada
IBM Yamato

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 235 to the fax number shown on the form.
- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Send your comments in an Internet note to redbook@us.ibm.com

Part 1. The AS/400 PASE runtime environment

This part is intended for use by all who are interested in AS/400 PASE. This may include interest in porting existing applications or understanding AS/400 PASE sufficiently to manage applications purchased from a software provider.

Chapter 1. Introduction to the AS/400 PASE runtime environment

AS/400 Portable Application Solutions Environment (AS/400 PASE) is designed to expand the AS/400 platform solutions portfolio by allowing customers and software vendors to port existing AIX applications to the AS/400 system with minimal effort.

AS/400 PASE is an integrated runtime environment for AIX (or other UNIX-like) applications running on the AS/400 system. It provides a broad subset of the Application Binary Interface (ABI) of AIX 4.3.3. As a real runtime (and not an emulator), it does not suffer the drawbacks of an emulation environment. However, AS/400 PASE is not a UNIX OS on the AS/400 system, or is it Linux on the AS/400 system. AS/400 PASE is designed to accept direct ports from AIX. Ports from any other UNIX-based environment may require an initial port to AIX as the first step towards compatibility.

AS/400 PASE is an optional, nominally priced feature of OS/400 (option 33), which requires AS/400 e-series processor-based machines or newer to run. V4R5M0 AS/400 PASE ships with a subset of the AIX Version 4.3.3 shared libraries. See 3.2, “Licensing issues” on page 33, for points to consider if you use other AIX libraries.

This redbook explains how to port an AIX application to AS/400 PASE at V4R5 only. Earlier and later releases of OS/400 or AS/400 PASE may have significantly different capabilities and limitations.

1.1 Executive overview

The AS/400 platform is designed for business. It leads the industry with its extensive industry-specific applications portfolio, which targets sectors such as banking and finance, industrial, retail and wholesale distribution, insurance, and telecommunications. The AS/400 system’s traditional strengths in core business and online transaction processing (OLTP) application markets set the stage for its leadership in midrange Enterprise Resource Planning (ERP) markets. More recently, the AS/400 solutions portfolio has expanded rapidly as new application markets emerged for e-commerce, groupware, supply chain management, customer relationship management, and business intelligence.

The AS/400 system's broad base of applications is continually enhanced by new applications coming to the platform from a variety of sources. Up to now, the AS/400 integrated language environment (ILE) accounted for the majority of C or C++ application ports, many of which originally ran on UNIX. Recently,

however, the AS/400 system focused on investment in Java and Domino (and even Windows NT on the Integrated Netfinity Server), which has opened up wide new application porting and modernization opportunities for solutions developers. With the latest addition of the AS/400 Portable Application Solutions Environment (AS/400 PASE), solutions developers have another option for porting UNIX applications rapidly to take advantage of the AS/400 systems extensive marketplace.

IBM's strategy for AS/400 PASE (Figure 1) is to expand its solutions portfolio, focusing on specific industry and application segments. One example is new supply chain management solutions that integrate with ERP applications targeted at industrial and distribution industries.

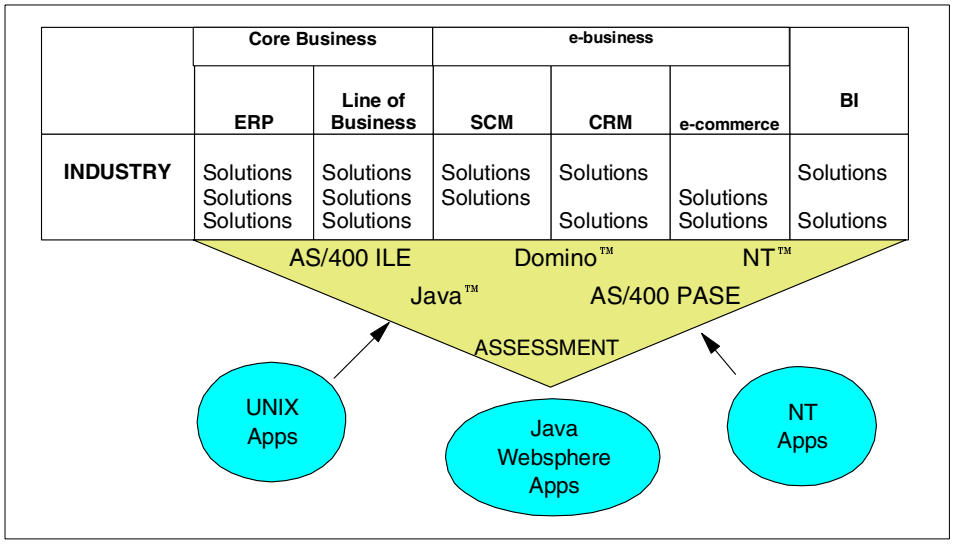


Figure 1. AS/400 PASE's place in the IBM strategy

Not all porting solutions on the AS/400 system equally apply to all applications. Certain applications will fit better in ILE while others fit better in AS/400 PASE. It is not our goal to push you in a particular direction with this book. Rather, it is our goal to show you how you can port an appropriate application into AS/400 PASE.

1.1.1 Positioning of AS/400 PASE

AS/400 PASE is not a replacement for AIX, a Linux environment on the AS/400 system, or a replacement for ILE or OS/400. It is another application enablement option on the AS/400 system.

Customers running an AS/400 PASE application may not need UNIX or AIX skills. Operationally, AS/400 PASE applications look just like any other AS/400 application. In cases where UNIX scripts need to be modified to tailor the application for the customer's needs, the systems integrator may require minimal UNIX skills.

AS/400 PASE adds another porting option for solutions developers who want to share in the success of the AS/400 marketplace. By providing a means to cut porting time significantly, AS/400 PASE can improve the time to market and return on investment for solutions developers. The AS/400 system attracts customers who want an integrated system that is built for business, offering broad application solution choices for their industry. AS/400 PASE is for companies who want to run UNIX applications without the complexity of running UNIX systems, and in a manner that is consistent with the AS/400 system's integrated value proposition and low cost of ownership.

AS/400 PASE provides an application runtime that is based on a broad subset of AIX technology. Applications are developed and compiled on an AIX workstation running a level of AIX that is compatible with a level supported by AS/400 PASE, and then run on the AS/400 system. AS/400 PASE exploits IBM's investment in a common processor technology for the RS/6000 and AS/400. The PowerPC processor switches from its normal AS/400 mode into RS/6000 mode to execute an application in the AS/400 PASE runtime. But the two brands serve separate markets. The AS/400 system sells to customers who want an integrated system that is built for business. The RS/6000 sells to customers who want to run the leading commercial UNIX system.

AS/400 PASE is supported on all AS/400e series servers (AS/400 systems introduced on or after September 1997). This means that some customers or software providers may need to upgrade their hardware to take advantage of AS/400 PASE.

Applications running in AS/400 PASE may need to be enabled to access DB2 Universal Database for AS/400 or integrated with AS/400 security and operations, such as backup.

The support for direct porting of application binaries is limited, depending on the APIs used by the application. Some binaries will run without change, while others may require limited or substantial modifications.

Application providers can obtain more support for porting their applications to AS/400 PASE from PartnerWorld for Developers at:
`http://www.ibm.com/as400/developer/factory/`

1.1.2 Planning a port to AS/400 PASE

When you are planning to port an application from AIX to the AS/400 system, you have two choices: you can port to the AS/400 ILE environment or you can port to the AS/400 PASE environment. Appendix C, “AS/400 PASE compared to ILE” on page 211, has tables that compare ILE and AS/400 PASE. Some of the strongest aspects of AS/400 PASE are outlined here:

- AS/400 PASE has a large number available operating system services. If the UNIX/AIX APIs you use are already supported, there is very little application porting to do. You can determine how your application stacks up against AS/400 PASE’s supported APIs by using PartnerWorld for Developers API Analysis process. The frontend tool in the process can be found at `http://www.ibm.com/as400/developer/porting/apitool.html`

  This is a shell script that produces a list of APIs that you send to the IBM porting team. The IBM team runs the backend analysis and returns a report to you indicating whether the application fits well on the system and where modifications, if any, may be needed. More information on this process is in Chapter 5, “Application API analysis” on page 45.
- AS/400 PASE provides a good environment for running computationally intensive applications on the AS/400 system by providing highly optimized math libraries.
- AS/400 PASE allows you to use AIX-based build processes, which is especially useful when you have an existing, complicated process that is not readily movable to a new platform. This would include testing harnesses built to verify a product after its final build.
- AS/400 PASE is tailored so that existing AIX applications will find porting to the AS/400 PASE environment simple and friendly.
- AS/400 PASE supplies support for `fork()` and `exec()`, which do not currently exist on the AS/400 system (except through `spawn()`, which can be somewhat different).
- AS/400 PASE is currently the best tool for satisfying dependencies on an ASCII character set and for satisfying dependencies on X-Windows support.
- Applications that rely on a large number of pointers and pointer casting may have an easier port to a 32-bit AIX addressing model.
- AS/400 PASE fully supports several dialects of C, C++, and Fortran.
- Shell programming is well supported.

There are a number of steps you need to follow when planning and executing a port to AS/400 PASE:

1. Obtain a complete copy of the application, with both source code and object code.
2. Obtain an API analysis from IBM to look for trouble spots.
3. Port the application to AIX Version 4.3.3 if it hasn't been ported there already.
4. Obtain all the hardware and software pre-requisites for running AS/400 PASE (an AS/400e with V4R5 and option 33).
5. Fix unsupported APIs in the source code, adding ILE calls or CLI library use for the AS/400 system if necessary, and recompile new binaries on the AIX system.
6. Move the binaries and datafiles to the AS/400 system, and run the system using "launchers" and environment variables to work around missing syscalls.
7. Iterate between edits and recompiles on the AIX system and test the AS/400 PASE environment until the application works.
8. Complete productization work, as covered in Appendix B, "The Application Factory" on page 205. Applications delivered to run AS/400 PASE can be wrappered and packaged so that they look and feel exactly like native AS/400 applications from an operations perspective.

1.2 Technical overview

AS/400 PASE is an integrated OS/400 runtime for porting AIX (and other UNIX) applications to the AS/400 system. To accomplish these ports, AS/400 PASE uses a subset of AIX runtime libraries. However, it is not an operating system or an emulated environment.

AS/400 PASE uses the AS/400 processor's ability to switch runtime modes to enable running AS/400 PASE applications concurrent with AS/400 ILE applications. In fact, AS/400 PASE is intimately integrated with the ILE environment, the AS/400 file systems, and DB2 Universal Database for AS/400. See Figure 2 on page 8 for a graphical depiction of the AS/400 PASE architecture. It can easily call Java (via a thin wrapper) and AS/400 ILE applications and, therefore, exploit all aspects of an AS/400 operations environment.

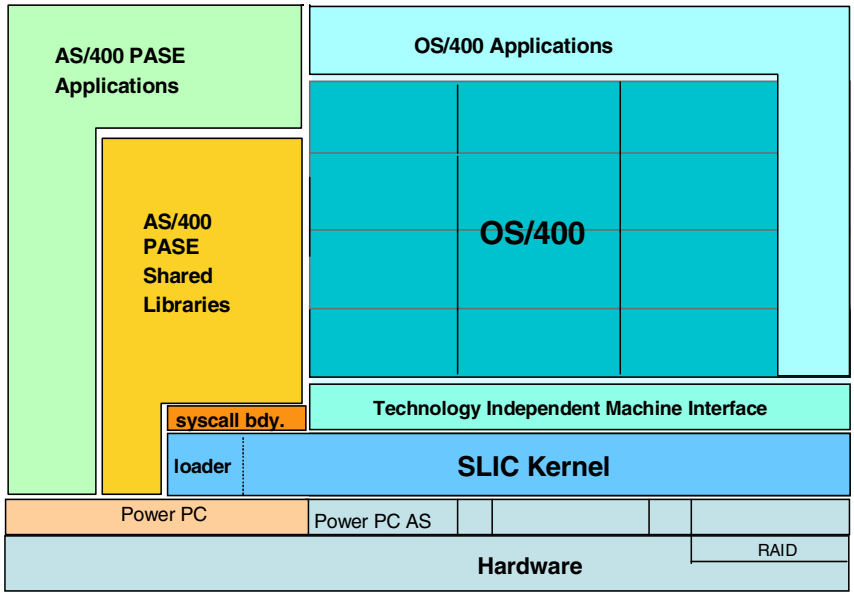


Figure 2. AS/400 PASE combine architecture (simplified)

The RS/6000 and AS/400 system share a common PowerPC chip. This hardware base has the ability to switch between runtime modes: addressing tags active to execute AS/400 64-bit applications and addressing tags inactive to execute 64- or 32-bit AIX applications. AS/400 PASE exploits this switching capability to execute 32-bit AIX applications on the AS/400 system within AS/400 jobs. Since AS/400 PASE applications execute directly on the hardware in PowerPC mode, computationally intensive applications use the processor without any additional layers overhead. In addition to AS/400 PASE accessing the Power PC instruction set on the AS/400 system directly, it can access the AS/400 PASE shared libraries (which are the ported AIX libraries), or, from the shared libraries, it can access a syscall interface to the SLIC Kernel. AS/400 PASE can also call out into the OS/400 ILE environment and access ILE applications and DB2 UDB (Universal Database) for AS/400.

The System Licensed Internal Code (SLIC) kernel controls the use of the hardware and the type of address space that can be used. SLIC is a common kernel under the AS/400 PASE shared libraries and OS/400. The services that SLIC provides are common to both the integrated language environment (ILE) and AS/400 PASE environments, creating a well-integrated system, regardless of which environment the application uses.

The environments share the same file systems, security mechanisms, and threading mechanisms. They can even share memory and connect to the same sockets (with a little care to note the pointer format expectations and the character set being shared).

SLIC and OS/400 work together to provide an object-based OS on the AS/400 system. AS/400 PASE has not caused the creation of new system objects. It uses the ones that were already built to support standard APIs for ILE. This facilitates sharing between application code and also gives AS/400 PASE an operational view that is just what AS/400 customers expect. To AS/400 operators, AS/400 PASE applications look like ILE or other running applications.

Programs and files for new AS/400 PASE applications are saved and restored in familiar ways, without the operator having to know that they run in a new way in operation.

As OS/400 sits architecturally on top of the Technology Independent Machine Interface (TIMI), AS/400 PASE sits architecturally on top of the syscall interface. AS/400 PASE looks architecturally like AIX in this respect. There is a protection layer between the kernel services for AS/400 PASE and non-privileged system or user code just as there is for OS/400. Therefore, the syscall is not designed to provide the same virtual machine as the TIMI. OS/400 has the capability to move applications that maintain their intermediate form to new hardware architectures (for example, CISC to RISC) without recompiling. This level of compatibility is not provided by the AS/400 PASE model.

AS/400 PASE applications are recompiled when the customer moves to a new hardware architecture, as one would expect with an AIX application making the same transition, since they have access directly to the hardware processor.

Applications that use AS/400 PASE are fully integrated in the customer's AS/400 workload and look no different operationally. AS/400 PASE can also call ILE functions, including application code and OS/400 system services. To an AS/400 programmer, AS/400 PASE looks like just another program model. To a UNIX programmer, AS/400 PASE provides the AIX Application Binary Interface (ABI) for their porting process.

Programs running in AS/400 PASE have direct access to the full capabilities of the user-state architecture of PowerPC, augmented by system services to interact with the Single-Level Store (SLS) environment. More information on how the addressing models can interrelate and share memory is provided in

8.1, “Shared addressing between AS/400 PASE and single level store” on page 100.

1.2.1 Hardware considerations

Most RISC processors used in AS/400 systems are designed to support both PowerAS and PowerPC architectures and provide the supervisor (SLIC) with the ability to switch architectures on demand. However, hardware implementation issues prevent us from using PowerPC mode on some older existing systems without jeopardizing user data or system integrity. AS/400 PASE is only supported on the AS/400e series (hardware introduced around September 1997). SLIC prevents AS/400 PASE programs from running on any system that is not AS/400e series hardware.

1.2.2 Development environment

AS/400 PASE development requires an AIX 4.3.3 system to run the compiler (xlc, xlC, or some other AIX compiler) and linker (ld command). The AIX assembler for PowerPC can also be used.

1.2.3 Performance

AS/400 PASE programs do not incur overhead to manage tagged pointers, MI boundaries, and the elaborate MI exception architecture, so low-level code generation may be superior to ILE in some cases. For example, incrementing a pointer can be done with a single instruction instead of the multi-instruction sequence required for MI, and a form of setjmp/longjmp that does not affect signal handling is provided that does not do any system calls.

Using a segment table for address translation increases the process working set and adds additional work in the hardware for address translation. However, the processors used in AS/400 systems contain Segment Lookaside Buffer (SLB) entries or segment registers to speed address translation. Address translation through an SLB entry or segment register is as fast as direct translation (used for tags-active mode), and AS/400 PASE programs should exhibit locality of reference similar to AIX (contributing to the effectiveness of SLB/segment register hardware).

Chapter 2. AS/400 PASE capabilities and limitations

This chapter introduces you to the basic advantages, capabilities, and limitations of AS/400 PASE.

2.1 Advantages of AS/400 PASE porting

In this section, we discuss the advantages of porting a UNIX or AIX application to AS/400 PASE.

2.1.1 Leveraging existing skills

AS/400 customers running applications ported using AS/400 PASE do not use UNIX system operations. AS/400 PASE applications run in AS/400 jobs using standard work management (subsystems), the AS/400 integrated file system with standard save/restore operations, and standard AS/400 security. No special system operations are required to run AS/400 PASE applications.

Independant Service Providers (ISVs) porting to the AS/400 system using AS/400 PASE and supporting their applications require a combination of AIX and OS/400 programming and support skills. AS/400 PASE applications are compiled on an AIX workstation. Primary application development continues using their existing UNIX skills where they have their greatest application value. AS/400 skills are required for converting database calls, providing integration with AS/400 security, and application operational interfaces (installation, startup, shutdown, and so on). These ancillary functions can often be built once and then maintained from release to release.

ISVs require AS/400 skills in their support teams since customers will call using AS/400, instead of UNIX, terminology.

2.1.2 Reducing time to market

Bringing an application to the AS/400 market requires three phases of effort:

- Market development
- Solution enablement (including porting the application)
- Market introduction

Market development and introduction are efforts common to all techniques of moving ISV applications to the AS/400 system. AS/400 PASE can improve several aspects of the solutions enablement phase. A faster port can mean a shorter time to market.

When using AS/400 PASE for enablement, some application ports require less effort since character set and pointer size assumptions do not change from what they are for AIX code. An ILE port requires the use of 128-bit ILE pointers and usually an EBCDIC character set. Also, AS/400 PASE runtime includes more flexibility in its C language support compared to the full ANSI compliant support provided in ILE C. A significant time savings may be achieved in the porting phase by not having to recreate the application's build environment. The testing cycle for an AS/400 PASE application may also be more similar to other UNIX platform testing.

There is a second part to enablement after the application is running. All applications, whether ported in ILE or AS/400 PASE, require a similar amount of customization to create an AS/400 product that meets AS/400 customers' expectations of full integration with database, save/restore, security, ease of installation, licensing, and robust support services. A product integration and test cycle is required for AS/400 PASE applications to fully exploit the AS/400 system environment. The Application Factory (see Appendix B, “The Application Factory” on page 205, for a full explanation of the Application Factory) was created to assist ISVs in identifying the items that should be examined for a new application to appeal to the AS/400 customer base.

2.2 Successfully ported applications

Table 1 lists the UNIX-based applications that are known to have been successfully ported to run in AS/400 PASE.

Note

The porting of the applications in Table 1 are not described in this or any other publication. It is left to the users of the application to perform the porting process, or contract through a service, such as the Rochester Custom Technology Center, to have the ports done for them.

These applications are not and will not be shipped with AS/400 PASE.

In Table 1, the license types are described as:

- **GPL**: GNU public license
- **LGPL**: Library GNU public license
- **NGPL**: Nethack GNU public license

Table 1. Successfully ported applications to AS/400 PASE

Package name	License	Where it can be found
ORBit	GPL	http://www.gnome.org/
Apache	Apache specific	http://www.apache.org/
gimp	GPL	http://www.gimp.org/
glib	LGPL	http://www.gtk.org/
gtk+	LGPL	http://www.gtk.org/
gnome-libs	GPL	http://www.gnome.org/
gnome-print	GPL	http://www.gnome.org/
gnumeric	GPL	http://www.gnome.org/
gtk-engines	GPL	http://www.gnome.org/
gzip	GPL	ftp://prep.ai.mit.edu/pub/gnu
icewm	LGPL	http://icewm.sourceforge.net/
imlib	GPL	http://www.gnome.org/
perl	GPL	http://www.perl.com/
libghttp	GPL	http://www.gnome.org
libglade	LGPL	http://www.gnome.org/
libxml	GPL	http://www.gnome.org
vnc	GPL	http://www.uk.research.att.com/vnc/
emacs	GPL	http://www.gnu.org/
gcc	GPL	http://www.gnu.org/
DBI perl module	GPL	http://www.perl.org/CPAN/modules/
Mysql perl module	GPL	http://www.perl.org/CPAN/modules/

Package name	License	Where it can be found
Chimera	GPL	http://www.cs.unlv.edu/chimera/
wget	GPL	http://www.gnu.org/software/wget/wget.html

2.3 Capabilities

AS/400 PASE utilizes many of the existing capabilities of OS/400.

2.3.1 File systems

All of the file systems available in OS/400 IFS are available within AS/400 PASE (see Table 2). See the integrated file system document listed in E.3, “Other resources” on page 222, for more information.

Table 2. File systems available to AS/400 PASE environment

File system	Description
/	Root file system
QOpenSys	Case sensitive, hierarchical file system; designed to support POSIX standards
QSYS.LIB	Library file system, library/file.member (database storage)
QOPT	Optical file system, CD-ROM access
QNTC	Windows NT servers using SMB, Microsoft’s file serving protocol
QFileSvr.400	OS/400 File Server, access to remote AS/400 systems
QDLS	Document Library Services, folder and document library objects; these were used by OV/400, the AS/400 office support product
/dev/QASPxx	User-defined File System, created in Auxiliary Storage Pool

2.3.2 Shared libraries

V4R5M0 AS/400 PASE provides a broad subset of AIX 4.3.3 functionality, such as standard C and C++ runtime (both thread safe and non-thread safe), Fortran runtime (both thread safe and non-thread safe), pthreads threading package, iconv services for data conversion, BSD equivalent support, X-windows client support with Motif widget set, and a database access library.

Table 3 lists the AS/400 PASE shared libraries that are shipped in OS/400 V4R5 option 33, and installed as symbolic links in /usr/lib.

Table 3. V4R5 AS/400 PASE shipped shared libraries

Library	Description
libC.a	C++ runtime
libC128.a	C++ 128-bit (type long double) runtime
libICE.a	Inter-Client Exchange library
libIM.a	Input method library
libMrm.a	Motif Runtime library for UIL
libSM.a	X Session Management library
libUil.a	Motif User Interface Language library
libX11.a	C interface for the X Window System protocol
libXaw.a	Athena Widget Set
libXext.a	Interfaces to X windows extensions
libXm.a	Motif widget library
libXmu.a	Miscellaneous X-windows utility functions
libXt.a	X Toolkit Intrinsics
libbsd.a	BSD Unix (TM) equivalence runtime
libc.a	C runtime
libc128.a	128-bit (type long double) runtime
libcrypt.a	C runtime cryptographic interfaces
libdb400.a	DB2/400 SQL CLI runtime
libdbx.a	dbx (debugger) utility support
libdl.a	Dynamic load runtime
libgaimisc.a	Internal-use X-windows support
libi18n.a	Internationalization runtime
libiconv.a	Character conversion runtime
libpthdebug.a	Threads debug support
libpthreads.a	Threads runtime

Library	Description
libpthreads_compat.a	Old threads compatibility
librtl.a	Runtime linking runtime
libxlf90_r.a	Fortran runtime
libxlfpthrds_compat.a	Old Fortran threads compatibility
libxlomp_ser.a	Open mp (multi-processing) support
libxlsmp.a	Symmetric mp (multi-processing) support

2.3.3 Runtime library members not supported

The runtime library members listed in Table 4 are either statically-bound or have dependencies that prevent them from loading in AS/400 PASE. All other members in PASE runtime libraries are shared executables that load without error (although they may contain symbols that are not fully-supported by AS/400 PASE). This list was accurate at the time this book was written. See the pre-GA note box on page ii of the edition notice.

Table 4. V4R5 AS/400 PASE runtime library exceptions

Library	Member	AS/400 PASE support	Comments
libbsd.a	__threads.o	No	Statically-bound (not shared executable)
libc.a	aio.o	No	Will not load in PASE (needs several syscalls)
libc.a	clc.o		Statically-bound (not shared executable)
libc.a	compi64.o		Statically-bound (not shared executable)
libc.a	compu64		Statically-bound (not shared executable)
libc.a	fill.o		Statically-bound (not shared executable)
libc.a	finite.o		Statically-bound (not shared executable)
libc.a	frexp.o		Statically-bound (not shared executable)

Library	Member	AS/400 PASE support	Comments
libc.a	fsavres.o		Statically-bound (not shared executable)
libc.a	fstab.o	No	Statically-bound (not shared executable)
libc.a	getttyent.o	No	Statically-bound (not shared executable)
libc.a	getttynam.o	No	Statically-bound (not shared executable)
libc.a	itrunc.o		Statically-bound (not shared executable)
libc.a	ldexp.o		Statically-bound (not shared executable)
libc.a	llabs.o		Statically-bound (not shared executable)
libc.a	logb.o		Statically-bound (not shared executable)
libc.a	longjmp.o		Statically-bound (not shared executable)
libc.a	modf.o		Statically-bound (not shared executable)
libc.a	moveeq.o		Statically-bound (not shared executable)
libc.a	move.o		Statically-bound (not shared executable)
libc.a	pse.o	No	Will not load in PASE (needs unsupported syscalls)
libc.a	pty.o	No	Will not load in PASE (needs _kgrantpt syscall)
libc.a	scalb.o		Statically-bound (not shared executable)
libc.a	strcat.o		Statically-bound (not shared executable)
libc.a	strcmp.o		Statically-bound (not shared executable)

Library	Member	AS/400 PASE support	Comments
libc.a	strcpy.o		Statically-bound (not shared executable)
libc.a	strncat.o		Statically-bound (not shared executable)
libc.a	strncpy.o		Statically-bound (not shared executable)
libc.a	uitrunc.o		Statically-bound (not shared executable)
libc.a	_itrunc.o		Statically-bound (not shared executable)
libc.a	_qint.o		Statically-bound (not shared executable)
libc.a	_qitrunc.o		Statically-bound (not shared executable)
libc.a	_qnint.o		Statically-bound (not shared executable)
libc.a	_quitrunc.o		Statically-bound (not shared executable)
libc.a	_setflm.o		Statically-bound (not shared executable)
libc.a	_uitrunc.o		Statically-bound (not shared executable)
libc.a	_xlqadd.o		Statically-bound (not shared executable)
libc.a	_xlqdiv.o		Statically-bound (not shared executable)
libc.a	_xlqmul.o		Statically-bound (not shared executable)
libc.a	_xlqsub.o		Statically-bound (not shared executable)
libc.a	__set_errno128.0		Statically-bound (not shared executable)
libpthreads.a	init.o		Statically-bound (not shared executable)

2.3.4 Runtime symbols

Some AIX symbols are either unsupported or have some support restrictions in V4R5 AS/400 PASE. A list of runtime symbols that have exceptions can be obtained from IBM PartnerWorld, AS/400 Software support, or the AS/400 PASE Web site located at:

`http://www.ibm.com/as400/developer/factory/pase/index.html`

Report any additions or corrections to IBM PartnerWorld or your local Software Support organization.

2.3.5 Shells and utilities

The default AS/400 PASE shell /QOpenSys/usr/bin/sh is the Korn shell. The Bourne and C shells are also available. The AS/400 system does not currently provide support for teletypewriter (tty) devices or Berkeley job control, so shell functions dependent on these elements are not supported by the AS/400 PASE shells.

The AS/400 PASE shells and utilities run in ASCII and do no conversion between ASCII/EBCDIC bytestream file data. Users can run the `iconv` utility to do conversions as needed.

The AS/400 PASE shells and utilities listed in Table 5 on page 20 are shipped with OS/400 option 33 (as symbolic links in directory /QOpenSys/usr/bin). AIX documentation (see the example that follows Table 5 on page 20) describes the syntax and behavior of all the shells and utilities listed above except for the AS/400-unique utility system, which provides an interface for invoking CL commands or programs from the AS/400 PASE terminal.

Table 5. AS/400 PASE supplied AIX utilities

alias	compress	expr	ln	pwd	true
apply	cp	false	locale	read	type
ar	cpio	fc	logname	rev	ulimit
awk	csh	fg	ls	rm	umask
banner	csplit	fgrep	mkdir	rmdir	unalias
basename	cut	file	mv	sed	uname
bc	date	find	nawk	sh	uncompress
bdiff	dbx	fold	newform	sleep	unexpand
bfs	dc	getconf	nl	sort	uniq
bg	dd	getopt	nm	split	unpack
bsh	diff	getopts	od	strings	untab
cat	diff3	grep	pack	strip	wait
cd	dircmp	hash	pagesize	sum	wc
chgrp	dirname	head	paste	system	what
chmod	dspcat	hostname	patch	tab	which
chown	dspmsg	iconv	pax	tail	xargs
chroot	du	id	pcat	tar	yes
cksum	dump	install	pr	tee	zcat
cmp	echo	jobs	printenv	test	
colrm	egrep	join	printf	time	
comm	env	kill	ps	touch	
command	expand	ksh	psh	tr	

The `system` utility is a unique AS/400 command that runs a CL command introduced in V4R5. The `system` utility manages ASCII/EBCDIC conversions for `stdin`, `stdout`, and `stderr` so that any ILE code run by the CL command uses EBCDIC data, while the AS/400 PASE shell and utilities see ASCII data. An explanation of the `system` utility is shown in the following sections.

Syntax

```
system [-b] [-h] [-i] [-k] [-K] [-n] [-q] [-s] [-v] CL-command
```

The system command runs a CL command. You may need to quote the CL command to avoid PASE shell processing for special characters in the command string.

Flags

-b Force binary mode processing for the `stdin`, `stdout`, or `stderr` files used by the CL command. When -b is not specified, the system command converts any data read from `stdin` from the (ASCII) PASE CCSID to the (EBCDIC) job default CCSID, and any data written to `stdout` or `stderr` from EBCDIC to ASCII.

This option only controls processing for stream data read and written by the CL command processing program. It does not affect the encoding of text lines written to `stdout` and `stderr` for messages and spooled output file data (which are always converted to ASCII).

-h Write a brief description of allowable syntax for the system command to `stdout`.

-i Run the CL command in the same process (OS/400 job) where the `system` utility runs.

Many CL commands are not supported in a multithreaded process. The `system` utility creates multiple threads to handle CCSID conversion for `stdin`, `stdout`, and `stderr`, so it defaults to running any CL command in a separate OS/400 job with only a single thread. Using the -i option can improve performance for CL commands that can tolerate operation in a multithreaded job.

-k Keep spooled output files after they are processed by writing the data to `stdout`. The `system` utility defaults to removing spooled output files produced by the CL command after it writes the data to `stdout`. This option retains the spooled output files.

-K Generate a job log for the process where the CL command runs.

In most cases, the `system` utility does not force a job log even if the CL command ends in error. *-K* can help problem determination when a CL command does not work as expected.

-n Do not include OS/400 message identifiers in any text line written to `stdout` or `stderr` for a message sent by the CL command.

The default format for any text lines written for OS/400 messages is "XXX1234: message text", where “XXX1234” is the OS/400 message identifier. -n suppresses the message identifier, so only the (first-level) message text is written to the stream.

-q Do not write any text lines to `stdout` or `stderr` for any OS/400 messages sent by the CL command.

-s Do not process spooled output files produced by the CL command. Spooled data is not written to `stdout`, and spooled output files are not deleted.

-v Write the CL command invocation string to `stdout` before running the CL command.

Exit status

0 The CL command completed successfully.

>0 An error occurred.

The example in Figure 3 shows both the power of the system command and the level of command integration. It runs the output through the `grep` command looking for a specific string.

```
                          /QOpenSys/usr/bin/sh

  $
> system dspusrprf spartz | grep -i spartz
    User Profile . . . . . . . . . . . . . . . :   SPARTZ
    Text . . . . . . . . . . . . . . . . . . . :   Daryl Spartz, PASE Gottfried

    Message queue  . . . . . . . . . . . . . . :   SPARTZ
    User Profile . . . . . . . . . . . . . . . :   SPARTZ
    Home directory . . . . . . . . . . . . . . :   /home/SPARTZ
  $




===>



F3=Exit   F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
F13=Clear F17=Top  F18=Bottom  F21=CL command entry
```

Figure 3. Using both the command system and grep

2.3.6 Logging facilities

There are no direct logging facilities, such as the syslog daemon, in the AS/400 PASE runtime environment. Applications needing such services 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. See Chapter 8, “ILE integration with the AS/400 PASE environment” on page 99, and Chapter 12, “Problem determination and messages” on page 173, for more information.

2.3.7 OS/400 access

AS/400 PASE applications have the ability to call OS/400 ILE procedures through the _ILECALL() API (refer to 8.3.7, “_ILECALL() function” on page 129) and to run any CL command including the CL CALL command through

the systemCL() API (refer to 8.3.12, “systemCL function” on page 138). OS/400 system CL commands can be executed through the `system` utility.

2.4 Limitations

In this section, we introduce the limitations of AS/400 PASE. As stated previously, the AS/400 PASE environment is a broad subset of the AIX runtime. The limitations imposed by this subset are described in the following sections.

2.4.1 Devices

The AS/400 PASE environment does not currently provide support for physical devices, including teletype (tty) or pseudo teletype (pty) devices. This coincides with no support of character or block special file system interfaces stated below.

The AS/400 system does not support direct access to ASCII terminal functions. ASCII devices attached to the workstation controllers can only be accessed using the 5250 data stream functions.

2.4.2 File systems

The AS/400 system does not currently support the creation of character special or block special files. The mknod and mkfifo APIs are not provided in AS/400 PASE. Also, the AS/400 system does not support linking application-written file systems into the kernel.

If your application requires a case sensitive file system, you need to put all your files into /QOpenSys, which is the only case sensitive file system provided by OS/400. However, any-user defined file system can be case sensitive.

Files in AS/400 PASE are always opened in binary mode. Therefore, no EBCDIC to ASCII conversion is performed, regardless of whether the file is tagged with a CCSID (`stdin`, `stdout`, and `stderr` can be exceptions to this rule). If needed, it is the responsibility of the application to provide the encoding translation. The `iconv()` system interface is available for this purpose.

There is a potential conflict of file system name space between OS/400 QSHELL and AS/400 PASE. The QSHELL environment reserves the /usr/bin part of the file system name space for its EBCDIC utilities. AS/400 PASE programs or scripts referring to that part of the file system need to point to

/QOpenSys/usr/bin. See 6.4, “Configuration tips” on page 62, for a possible workaround other than code modification.

2.4.3 Printing

There is no facility for printing directly from PASE. If the program is capable of writing ASCII data (for example PostScript) to an IFS stream file, the following OS/400 commands can be used to print the file. In this example, 200 is the maximum record length and the AS/400 PASE CCSID is 819:

```
CRTPF FILE(QGPL/PSPRT) RCDLEN(200)

CPYFRMSTMF FROMSTMF('/home/spartz/test.ps')
TOMBR('/qsys.lib/qgpl.lib/psprt.file/test.mbr') CVTDTA(*NONE)
STMFCODPAG(819) DBFCCSID(819) ENDLINFMT(*FIXED) TABEXPN(*NO)

OVRPRTF FILE(QPRINT) DEVTYPE(*USERASCII) PAGESIZE(*N 200)

CPYF FROMFILE(QGPL/PSPRT) TOFILE(QPRINT) FROMMBR(TEST)

LPR RMTSYS(PRTSYS) PRTQ('myPSprt') FILE(QPRINT) SPLNBR(*LAST)
TRANSFORM(*NO)
```

2.4.4 Shells

The AS/400 system does not currently provide support for Berkeley job control. Therefore, shell functions that depend on these elements are not supported by the AS/400 PASE shells.

2.4.5 Libraries

Only the libraries listed in 2.3, “Capabilities” on page 14, are supported. Other libraries, such as libdbm, libnsl, and libcurses are not provided. Do *not* copy libraries from an AIX system without first ensuring that you have a license to do so. See 3.2, “Licensing issues” on page 33, for more information.

2.4.6 API restrictions

Table 6 represents the current V4R5M0 list of restricted or flagged APIs, which the API analysis tool from PartnerWorld for Developers will identify. See Chapter 5, “Application API analysis” on page 45, for information on obtaining and using the tool. When referring to this list, you should be aware that the table used by the API Analysis tool changes based on changes or updates to the AS/400 PASE product. For the current table, see the

Table 6. AS/400 PASE flagged AIX APIs

API name	API type	Key text
_system_configuration	CLIB	Subset for kernel/hardware differences.
accessx	FILES	Only support ACC_INVOKER; or ACC_SELF if/when the effective and real uid/gid match.
closelog	SYSLOG	No syslog daemon; use AS/400 logging facilities instead. See 2.3.6, “Logging facilities” on page 22.
closelog_r	SYSLOG	No syslog daemon; use AS/400 logging facilities instead. See 2.3.6, “Logging facilities” on page 22.
creat	FILES	No S_ISVTX or S_ENFMT support.
creat64	FILES	No S_ISVTX or S_ENFMT support.
endfsent	CLIB	Use ILE interfaces for mounted file systems. See 2.3.7, “OS/400 access” on page 22.
endfsent_r	CLIB	Use ILE interfaces for mounted file systems. See 2.3.7, “OS/400 access” on page 22.
endttyent	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
endttyent_r	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
fcntl	FILES	Supported except F_CLOSEM.
fstatx	FILES	STX_XPFFD_PASE retrieves ILE descriptor number; STX_XPFSS_PASE retrieves ILE struct stat.
getgrent	AUTH	Returns ENOSUP; use getpwuid/getgrgid or getpwnam/getgrnam interfaces.
getgrnam/getgrgid	AUTH	Returns ENAMETOOLONG for member names longer than eight characters.
getitimer	CLIB	Only supports ITIMER_REAL.
getpgrp	PROC	Limited process group support without full Berkeley job control.
getpwent	AUTH	Returns ENOSUP; use getpwuid/getgrgid or getpwnam/getgrnam interfaces.

API name	API type	Key text
getpwnam	AUTH	AS/400 PASE always sets pw_gecos to zero.
getpwnam_r	AUTH	AS/400 PASE always sets pw_gecos to zero.
getpwuid	AUTH	AS/400 PASE always sets pw_gecos to zero.
getpwuid_r	AUTH	AS/400 PASE always sets pw_gecos to zero.
getrlimit	PROC	Supports RLIMIT_STACK only.
getrusage	PROC	Supports RUSAGE_SELF and RUSAGE_CHILDREN only.
getsockopt	SOCK	Partial support. See 2.4.6.2, “getsockopt/setsockopt support” on page 31.
gettimerid	CLIB	Only ITIMER_REAL supported.
getttyent	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
getttyent_r	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
getttynam	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
getttynam_r	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
getvfsbyflag	CONF	Use ILE interfaces for working with mounted file systems. See 2.3.7, “OS/400 access” on page 22.
getvfsbyname	CONF	Use ILE interfaces for working with mounted file systems. See 2.3.7, “OS/400 access” on page 22.
getvfsbytype	CONF	Use ILE interfaces for working with mounted file systems. See 2.3.7, “OS/400 access” on page 22.
getvfsent	CONF	Use ILE interfaces for working with mounted file systems. See 2.3.7, “OS/400 access” on page 22.
gtty	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
incinterval	CLIB	Only ITIMER_REAL supported.
ioctl	FILES	Partial support. See 2.4.6.1, “IOCTL support” on page 31.

API name	API type	Key text
ioctlx	DEV	Partial support; additional argument from ioctl ignored.
isatty	TTY	AS/400 PASE returns TRUE for the (non-tty) descriptors initially assigned to fd 0/1/2; environment variable PASE_STDIO_ISATTY=N overrides this behavior.
lseek	FILES	IFS restricts directories to sequential access.
lseek64	FILES	IFS restricts directories to sequential access.
mmap	SHM	Consider using shm... APIs instead.
mntctl	FILES	Use ILE interfaces for mounted file systems. See 2.3.7, “OS/400 access” on page 22.
mount	FILES	Use ILE interfaces for mounted file systems. See 2.3.7, “OS/400 access” on page 22.
mprotect	SHM	Partial support - does not work with shmxxx routines.
msem_init	MEM	Not supported.
msync	SHM	Consider using shm... APIs instead.
munmap	SHM	Consider using shm... APIs instead.
open	FILES	No S_ISVTX S_ENFMT O_DEFER O_SYNC O_DSYNC FAIO support; O_DELAY and O_NOCTTY ignored.
openlog	SYSLOG	No syslog daemon; use AS/400 logging facilities instead.
openlog_r	SYSLOG	No syslog daemon; use AS/400 logging facilities instead.
open64	FILES	No S_ISVTX S_ENFMT O_DEFER O_SYNC O_DSYNC FAIO support; O_DELAY and O_NOCTTY ignored.
privcheck	AUTH	EPERM except for SET_PROC_DAC SET_PROC_RAC and BYPASS_DAC.
psdanger	PROC	SIGDANGER returns system ASP storage threshold, but the AS/400 system does not send signals for storage limit problems.
ptrace	PROC	AS/400 PASE supports the parts of ptrace arch that AIX supports.

API name	API type	Key text
readvx	FILES	Partial support, additional argument from ioctl ignored.
readx	FILES	Partial support, additional argument from ioctl ignored.
semop	SEM	SEM_UNDO is thread-based versus process-based on AIX.
setegid	AUTH	Only allowed while AS/400 PASE is single-threaded.
seteuid	AUTH	Only allowed while AS/400 PASE is single-threaded.
setgid	AUTH	Only allowed while AS/400 PASE is single-threaded.
setgrent	AUTH	Returns ENOSUP; use getpwuid/getgrgid or getpwnam/getgrnam interfaces.
setitimer	CLIB	Only ITIMER_REAL supported.
setlogmask	SYSLOG	No syslog daemon; use AS/400 logging facilities instead. See 2.3.6, “Logging facilities” on page 22.
setlogmask_r	SYSLOG	No syslog daemon; use AS/400 logging facilities instead. See 2.3.6, “Logging facilities” on page 22.
setuid	PRIV	Only allowed while AS/400 PASE is single-threaded.
setpenv	AUTH	The AS/400 system does not have session support.
setpgrp	PROC	The AS/400 system does not have session support.
setpwent	AUTH	Returns ENOSUP; use getpwuid/getgrgid or getpwnam/getgrnam interfaces.
setregid	AUTH	Only allowed while AS/400 PASE is single-threaded.
setreuid	AUTH	Only allowed while AS/400 PASE is single-threaded.
setsockopt	SOCK	Partial support. See 2.4.6.2, “getsockopt/setsockopt support” on page 31.
setttyent	TTY	Use ILE interfaces for working with mounted file systems. See 2.3.7, “OS/400 access” on page 22.
setttyent_r	TTY	Use ILE interfaces for working with mounted file systems. See 2.3.7, “OS/400 access” on page 22.
setvfsent	CONF	Use ILE interfaces for mounted file systems. See 2.3.7, “OS/400 access” on page 22.
shmat	SHM	shmat of a file descriptor not supported.

API name	API type	Key text
statvfs	FILES	Use ILE interfaces for mounted file systems. See 2.3.7, “OS/400 access” on page 22.
statx	FILES	STX_XPFSS_PASE retrieves ILEstat structures.
stty	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
sysconf	PROC	Partial support. Only SYS_GETPARMS is supported with fields v_ncpus_cfg and v_ncpus filled in.
sysconfig	CONF	Subset (SYS_GETPARMS) support.
syslog	SYSLOG	No syslog daemon; use AS/400 logging facilities instead. See 2.3.6, “Logging facilities” on page 22.
syslog_r	SYSLOG	No syslog daemon; use AS/400 logging facilities instead. See 2.3.6, “Logging facilities” on page 22.
system	CLIB	Please examine source code to see what is called; check utilities lists for the AS/400 system as needed.
tcdrain	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
tcflow	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
tcflush	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
tcgetattr	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
tcgetpgrp	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
tcgetsid	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
tcsendbreak	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
tcsetattr	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
tcsetpgrp	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.

API name	API type	Key text
telldir	FILES	AS/400 PASE seekdir not recommended. Do not position an open directory to anything other than the beginning.
termdef	DEV	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
thread_setsched	THREADS	Stubbed to no-op.
time	PROC	AS/400 PASE does not round up to next second (ILE does).
times	PROC	Subset support (only user time).
ttylock	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
ttylocked	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
ttyname	TTY	AS/400 PASE always return NULL (ENOTTY).
ttyname_r	TTY	AS/400 PASE always return NULL (ENOTTY).
ttyslot	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
ttyunlock	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
ttywait	TTY	Some apps use isatty() to avoid other tty use if unneeded; presence does not always cause failures.
umount	FILES	Use ILE interfaces for mounted file systems.
wait3	PROC	Supports RUSAGE_SELF and RUSAGE_CHILDREN only.
writevx	IO	Partial support, additional argument from ioctl ignored.
writex	FILES	Partial support, additional argument from ioctl ignored.
SQLGetSubstring	CLI	Always returns an EBCDIC substring in CLOB/DBCLOB data types.
Trconflag	DEBUG	Stubbed to no-op.

2.4.6.1 IOCTL support

The AS/400 PASE environment provides a limited set of supported options for the ioctl API. Those supported in OS/400 V4R5 are listed in Table 7.

Table 7. AS/400 PASE support of ioctl API

FIOASYNC	FIOSETOWN	SIOCGIFFLAGS
FIOCLEX	ISATTY	SIOCGIFMTU
FIOGETOWN	SIOCATMARK	SIOCGIFNETMASK
FIONBIO	SIOCGIFADDR	SIOCGPGRP
FIONCLEX	SIOCGIFBRDADDR	SIOCSPGRP
FIONREAD	SIOCGIFCONF	

2.4.6.2 getsockopt/setsockopt support

The AS/400 PASE environment provides a limited set of supported options for the getsockopt() and setsockopt() APIs. Since AS/400 PASE shares the TCP/IP stack with OS/400, the same restrictions occur in OS/400. The APIs that are supported are listed in Table 8.

Table 8. AS/400 PASE support of getsockopt/setsockopt APIs

SO_BROADCAST	SO_OOBINLINE	SO_SNDBUF
SO_DEBUG	SO_RCVBUF	SO_SNDLOWAT
SO_DONTROUTE	SO_RCVLOWAT	SO_SNDTIMEO
SO_KEEPALIVE	SO_RCVTIMEO	TCP_NODELAY
SO_LINGER	SO_REUSEADDR	

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.

2.4.7 Unsupported system calls

The AS/400 PASE kernel exports some system calls that are implemented by the AIX kernel but are not supported in AS/400 PASE. Details of how these system calls behave and how to debug them can be found in 6.6.2, “AS/400 PASE unsupported system calls” on page 85.

Chapter 3. Getting started with AS/400 PASE

AS/400 PASE has several hardware and software prerequisites. These prerequisites change depending upon whether you are an end user of a ported AS/400 PASE application or if you are porting an application from AIX to AS/400 PASE.

3.1 Requirements

All users are required to have the following requirements:

- OS/400 V4R5 or later (AS/400 PASE will run on V4R4, but is more limited in support and is not covered in this book)
- Option 33 of the OS/400 (5769SS100)
- Current PTFs. A list of current PTFs can be found at:
  `http://www.ibm.com/as400/developer/factory/pase/misc.html`

  If this page doesn't show the V4R5M0 PTFs for AS/400 PASE, contact your local AS/400 Software Support organization.

People porting code from AIX are required to have:

- RS/6000 running AIX 4.3.3
- PC running an X-Windows server host from an AIX 4.3.3 box. Several commercial products exist, including Hummingbird Exceed (`http://www.hummingbird.com/`) and X-Win32 (`http://www.starnet.com/productinfo/`)

Most RISC processors used in AS/400 systems were designed to support both PowerAS and PowerPC architectures and provide the ability for the supervisor System Licensed Internal Code (SLIC) to switch architectures on demand. However, hardware implementation issues prevent us from using PowerPC mode on some older systems without jeopardizing user data or system integrity. AS/400 PASE is formally supported on the AS/400e Series (hardware introduced around September 1997). SLIC prevents AS/400 PASE programs from running on any system that is not AS/400e Series hardware.

3.2 Licensing issues

AS/400 PASE provides a subset of the AIX runtime libraries on the AS/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 AS/400 PASE. All AIX products are separately licensed by IBM.

As you begin porting your own applications to AS/400 PASE, you may find that your application has dependencies on AIX libraries that were not shipped with AS/400 PASE. Before porting these libraries to the AS/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 AS/400 system.

We strongly encourage you to 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. See Appendix E.4, “Referenced Web sites” on page 222, for a list of Web sites.

3.3 Recommendations

People that port AIX code to AS/400 PASE need to closely consider these guidelines:

- Networking between OS/400 and AIX machines
- Knowledge of both OS/400 and AIX or UNIX

  Many ports do not require an individual who has extensive knowledge in both environments. However, individuals who have deep knowledge in each of the areas are useful. OS/400 knowledge is most needed when you wrapper your product or need to execute a database port.
- AS/400 PASE was developed to target AIX 4.3.3. However, code created on AIX 4.3.X releases is very likely to be upward compatible and, therefore, should be fully portable.

3.4 General AS/400 PASE instructions

The following section lists and generally describes the steps you may go through if you are porting a solution to AS/400 PASE.

3.4.1 General system setup

On your e-series AS/400 system, install OS/400 V4R5 with option 33. This option is specifically orderable from IBM and is a chargeable feature of OS/400.

Check for any AS/400 PASE PTFs that were created since the installation level of your software and load those too.

You need to have an AIX system available for compilation. It will be at the AIX 4.3.3 level, preferably, but most AIX 4.3 generated code will work since AIX offers good upward compatibility.

3.4.2 Checking system support

It is often useful to run an API analysis on the solution to be ported. This allows you to check the match between the APIs supported by AS/400 PASE and those that the solution uses. This process is usually a small task and can speed the prototyping stage greatly. See Chapter 5, “Application API analysis” on page 45, for more information.

3.4.3 Preparing the application

You need to be sure that your AS/400 PASE program is compiled correctly on AIX, especially if you depend on calling ILE routines or database. See the instructions in 6.2, “Compiling applications on AIX for AS/400 PASE” on page 58, for more information. Many programs that are standalone will work correctly as compiled for AIX.

AS/400 PASE programs are not “called” directly from the command line. You need to either run one of the “launcher” programs, QP2TERM or QP2SHELL, or write an ILE C program to call Qp2RunPase. If this is your first experience with AS/400 PASE, and you are not familiar with ILE C compilations, we recommend that you use the “launchers”. The launchers are tools intended for prototyping. See the detailed usage documentation in 6.5, “Starting an AS/400 PASE application” on page 65, for more information. Users of formal releases of PASE applications most often want to write the C programs or their own launchers to create the production environment they need.

3.4.4 Testing the application

AIX 4.3 binaries that are to be run must be moved to the root of the stream file system “/” on the AS/400 system. If your application is sensitive to mixed case, move them into the /QOpenSys file system under the root. Create the path as appropriate.

Use FTP to make sure you are using NAMEFMT 1 so that the files will be transferred as stream files into stream files. If you have trouble getting the parameter typed in and you are looking for an alternative, or if the first command you type into FTP uses an AS/400 file name that starts with a slash (for example, /dir/myfile), enter the NAMEFMT 1 mode automatically by the ftp command.

3.4.5 Creating your AS/400 solution

Once your solution is running, examine some of the typical AS/400 customer expectations to see what additional application parts would give them a familiar look and feel. These topics are covered in Appendix B, “The Application Factory” on page 205.

Part 2. AS/400 PASE from a UNIX perspective

Part 2 is intended for users who will port a UNIX or AIX application to run in AS/400 PASE.

Chapter 4. AS/400 architecture from a UNIX perspective

This chapter discusses certain architectural characteristics of the AS/400 system and the integrated language environment (ILE). The aspects considered are those of interest to a UNIX C application developer. This chapter also discusses the characteristics of a UNIX operating system as they relate to the AS/400 computer architecture.

AS/400 system architecture is defined by a fairly high-level machine interface (MI), sometimes referred to as a Technology Independent Machine Interface (TIMI). “Technology Independence” refers to the fact that the applications (and much of the operating system) have no dependence on the type of processor, whether it's Complex Instruction Set Computer (CISC), Reduced Instruction Set Computer (RISC), Very Long Instruction Words (VLIW), and so on. This section discusses the following architectural aspects of the AS/400 system:

- Object-oriented architecture
- Addressing or storage management
- Contexts (AS/400 libraries) and address resolution
- User profiles and authority management
- Character sets and terminal I/O

The particular emphasis is on distinctions from what you might expect of an architecture supporting a UNIX operating system.

4.1 Object-oriented architecture

Objects are the means through which information is stored and retrieved on the AS/400 system. This concept is different from the typical byte-stream file manipulation on many systems. Object orientation is part of the architecture and affects both operating system implementation and high level language interaction with the system.

As previously mentioned, the MI is a boundary (set of instructions) that separates the hardware and System Licensed Internal Code (SLIC) from the operating system. SLIC contains traditional kernel-type functions, such as storage management, resource management, authority checking, and so on.

There are very fundamental objects in SLIC that are put together to create complex objects in OS/400 above the MI. For example, a cursor object, a physical file object, and several other objects are used to implement a relational database table in OS/400. Each object has operational

characteristics and a defined set of operations that can be performed on it. Objects are addressed through 16-byte pointers (8 bytes are used for an MI address; the other 8 bytes are used for information about the object pointed to and for reserved space). In addition to providing addressability to the object, pointers provide access to the associated storage, data integrity, and security. Above the MI, the contents of the pointer are encapsulated or protected. For example, the hardware itself prevents an application from fabricating a pointer from an integer.

Below the MI, the AS/400 hardware provides a logical tag bit for each quadword (16 bytes that must be aligned on a 16-byte boundary) within main storage. This bit is not accessible by the normal application instructions used to access storage. The bit, when it is equal to 1, identifies quadwords in storage containing MI pointers. Programs above the MI have no direct access to the tag bit. The tag bit is turned on by SLIC when a pointer is set and turned off by the hardware any time the quadword is modified (except through a controlled set of SLIC pointer manipulation instructions). This procedure allows the system to detect invalid pointers and prevent illegal use of a pointer. Any attempt to subsequently use this data as a pointer results in an exception and the instruction is not completed. It is not possible to counterfeit a pointer or to modify a pointer in an invalid way.

The tag bit implementation allows the validation of pointers in an extremely efficient way. It is the basis for system and data integrity since pointers can contain authorization information as well as addresses.

Implications for High Level Language, including pointer arithmetic, assignment operations, casting, compares, and so on, all work as expected as long as the system is aware that variables are address pointers. A pointer in the ILE C/400 is 16 bytes long. Programming tricks, such as manipulating pointers in integer-sized space, do not work.

On a 32-bit UNIX system, integers and pointers are both typically 4 bytes long. An integer can be cast to a pointer. An address of an object of one data type can be assigned to a pointer of another data type without proper pointer casting. On the AS/400 system, such assignment operations produce exceptions.

4.2 Addressing and storage management

AS/400 storage management uses single level storage. With single level storage, there is a single, large, uniformly addressable address space for all memory (both main storage (RAM) and secondary storage (DISK)). Storage

is addressed by a 64-bit address. This large virtual address equates to roughly 18 million terabytes of addressable storage.

There is a single page directory that maps all virtual addresses to the corresponding physical addresses. Addresses are unique across the system for the life of the system and are not duplicated across processes. The same address in a different process points to the same storage location. This concept is different from UNIX (where there is one address space per process). It also has implications for how storage is managed and how processes are created and managed.

The UNIX System V kernel divides the virtual address space of a process into logical regions. A region is a contiguous area of the virtual address space that can be treated as a distinct space to be shared (with other processes) or protected. UNIX address spaces are per process. The same address in different processes can point to different memory spaces. Since the AS/400 address space is per system, the same address in different processes always points to the same memory space. Consequently, the way addresses are translated and the way memory is managed is fundamentally different between the AS/400 architecture and from that which is typically associated with UNIX systems. These differences are summarized in Table 9.

Table 9. AS/400 and UNIX storage management differences

AS/400 system	UNIX
Single level storage	Process-based addressing
Persistent addresses	Temporary addresses
Full job structure	Middleweight processes in a process group

PASE provides the UNIX model within the AS/400 model.

4.2.1 Job and process structure

An example of a UNIX system call that is very expensive to implement on the AS/400 in ILE is `fork()`, which is how a process is created on a UNIX system. The UNIX kernel does (among other things) the following operations for `fork()`:

- Allocates a slot in the process table for the new process
- Assigns a unique ID number for the child process
- Makes a logical copy of the parent process

The notion of copying storage (which contains pointers) of the parent process is inconsistent with the AS/400 single level storage architecture. On UNIX systems, pointers are relative to the process. On the AS/400 system, pointers are absolute because of the single address space for the entire system. Consequently, `fork()` (in UNIX terms) is not currently implemented on the AS/400 system outside the PASE environment. A combination of `fork()` and `exec()` semantics has been implemented on the AS/400 system with `spawn()` and related APIs. In addition, the AS/400 dynamic call allows an ease-of-use call for multiple “main” programs while remaining in the same process. This provides single-thread dynamic execution in which programs can work together in a single security and resource environment that is not available in UNIX.

4.3 Library and address resolution

At the time an AS/400 object is created, the operating system places the object name in (another) machine object called a *context*. Contexts are presented to the user as OS/400 libraries (not to be confused with UNIX libraries). The context object maps (resolves) the symbolic identification (type and name) of an object to its virtual addresses. A user-specified (and modifiable) list of libraries is associated with each job on the system. Objects can be referenced by the user and can be explicitly qualified to a specific library. If not explicitly qualified to a library, the library list of the job resolves the references by searching each library in the list in order until a matching entry is found. This resolution is roughly analogous to LIBPATH processing on an AIX system.

On UNIX systems, most objects are treated as a file and addressed through hierarchical directories. The AS/400 system has objects addressed through contexts (AS/400 libraries). This notion contrasts to the concept of symbolic name resolution as used with directories and file systems on UNIX systems.

The AS/400 system has a similarity with UNIX by supporting a hierarchical, case-sensitive, POSIX-compliant name space (called *QOpenSys*). Byte-stream files stored in QOpenSys are addressed through directories.

4.4 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 QOpenSys are authorized in the same manner as UNIX files. Figure 4 on page 43 shows the relationship between UNIX permissions and security used on AS/400 database files.

Mapping UNIX Permissions to AS/400 Security

UNIX	AS/400					
	*OBJOPR	Data Authority				
		*READ	*ADD	*UPD	*DLT	*XEQ
r(read)	X	X	-	-	-	-
w(write)	X	-	X	X	X	-
e(execute)	X	-	-	-	-	X
No Authority	-	-	-	-	-	-

(*OBJOPR-Use object, *EXCLUDE- No Authority)

Figure 4. Mapping UNIX permissions to AS/400 security

4.5 Character sets and terminal I/O

This section describes the differences between two architectures, such as character sets and terminal I/O.

4.5.1 EBCDIC versus ASCII

Most UNIX systems historically run on hardware that uses ASCII character encoding. The AS/400 system uses EBCDIC encoding by default, although all objects can be tagged with specific code pages. This architectural difference is not a problem, assuming high level language applications do not have a dependency on (or make an assumption about) the character set. Problems arise if, for example, applications are coded with dependencies on a hexadecimal representation of character. The hexadecimal representation varies between ASCII/EBCDIC. Similarly, the collating sequence (that is, ordering of characters) also differs.

4.5.2 Buffered versus unbuffered I/O

Input and output to and from external devices is buffered on the AS/400 system. I/O is handled by I/O processors that deal with blocks of data. Conversely, UNIX systems typically operate with character-by-character (unbuffered) I/O. On the AS/400 system, only certain I/O signals (for example, the Enter key, function keys, and system request) send an interrupt to the CPU.

4.5.3 I/O controllers versus device drivers

It may not be common, but it is possible, on most UNIX systems (as well as most personal computers) to write applications that include device drivers for devices specific to an application. That is, it is possible to write applications utilizing a port that controls an I/O device. On the AS/400 system, I/O is handled by system I/O managers (IOMs) that communicate with device controllers to handle I/O requests.

Since IOMs are part of the AS/400 architecture, applications do not write directly to a particular device. Also, linking application code into the kernel (SLIC) is not allowed.

4.6 Architectural summary

This chapter has discussed AS/400 architectural distinctions from UNIX. These differences should not affect or impede source code portability of applications to the AS/400 system (assuming that the code is written in an otherwise portable manner).

Significant architectural features that uniquely identify the AS/400 system include high-level machine interface (which really defines the architecture), object-orientation, and single-level storage. The high-level machine interface permits the underlying implementation of the hardware to change without affecting users above the MI (including the operating system and enduser applications). Further, these architectural features inherently provide a high degree of data and system integrity, authorization, and reliability. These features, if they exist on a UNIX system, typically must be provided by higher-level functions of the operating system or by the application.

Chapter 5. Application API analysis

This chapter discusses, in depth, how to analyze your application to see how well it matches the APIs that AS/400 PASE supports.

5.1 Introduction

API analysis is the first step in the feasibility study and assessment of the portability of a UNIX C application to the AS/400 system. The API analysis filters the interfaces used within the application that are not industry standard and not supported on the AS/400 system. It also filters the interfaces that are standard compliant but supported differently due to the different architecture of the AS/400 system compared to UNIX machines.

The API analysis tool consists of a frontend and a backend. The frontend scans the executables of the application to extract the interfaces (or function calls) used by the application and generates a list of all those interfaces. The backend of the tool takes this list of interfaces as input and compares them with a database of “typical” system APIs and their support.

The frontend of the API analysis tool is a UNIX shell script and uses the `nm` or `dump` command to find symbol information from the external symbol table of the executables. The shell script runs on one of the following operating systems, which is passed as a parameter to the script:

- AIX
- HP-UX
- SCO UNIX
- SUN Solaris
- Digital UNIX

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

5.2 What you need to do

You need to download the API Analysis Tool Frontend, which is located on the Web at: `http://www.ibm.com/as400/developer/porting/apitool.html`

Choose your own browser's option to save the API Analysis Tool Front-end to your local disk in a file (for example, file rtvapi). Change the file mode to *executable* with the following command:

```
chmod +x rtvapi
```

Run this script against the executables of your application. You need to supply an operating system and (optionally) a file or directory name as parameters:

```
rtvapi {AIX|HPUX|SunSolaris|ScoUNIX|DigitalUNIX} [<file name>|<directory
name>]
```

While running the tool, make sure:

- The executables of the application are not stripped of symbol table information.
- You have write permission in the current directory since the tool will create some temporary files.
- When a file name is supplied, it must be an executable file with read permission.
- If a directory name is supplied, it must be a searchable directory. The tool drills down to all the subdirectories to find all the executable files starting from the directory specified.
- You redirect the output (which is a list of function calls) to a file.

The IBM porting team in PartnerWorld for Developers will run the backend of the API analysis tool. Send the output file and a short description of your application and target marketplace via e-mail to `rchgo400@us.ibm.com`. Also, if you have any problems using the shell script, send an explanatory e-mail to the same address.

5.2.1 API analysis example

The following example report analysis uses the porting example of OpenDX outlined in 10.2, “OpenDX” on page 150:

1. On the AIX machine, make a temporary directory. On the AIX machine, type:

   ```
   mkdir ~/tmp/opendx
   ```

2. Either FTP or copy all files from /usr/local/dx/bin_6000 and /usr/local/dx/lib_6000 to the temporary directory.
3. Run the API analysis tool. On the AIX machine, type:

   ```
   rtvapi AIX ~/tmp/opendx > ~/api.report
   ```

4. Submit the report to IBM as previously directed.

The following report summary is from IBM on OpenDX. The report generation does not cross-reference dependencies with all of the files on which the report is generated, that is user defined APIs. Therefore, there it is assumed that the submitter can recognize these APIs and manually remove them from the list.

Note that the analysis tool looks for system-related support and does not recognize C++ runtime calls since this is part of support.

Note

The full report is rather extensive, comparing the application API references to both ILE and AS/400 PASE in multiple releases. The following report excludes the sections that refer to ILE and release information prior to V4R5.

The report also shows as “unrecognized” APIs, APIs that the OpenDX exports in its own libraries. These APIs are numerous and are prefixed with DX, _dxt, _dxd, and _dxf. They have been removed.

IBM AS/400 API Analysis Report

Thank you for your interest in an AS/400 API Analysis. This file is your analysis report. While this report may not be a complete predictor of all port effort for your solution, we have found that this process does help us identify significant system support issues that need to be worked in porting activity. Please look through the details below for follow up technical items requiring further examination in your source code. Also, please note the percentages in the table below are simplified. All APIs in this report are weighted equally, irrespective of the number of times they are used in the solution. For example, if a solution used two APIs, one once and the other ninety-nine times, the support percentage would appear as 50%, while the actual system call support would be 99%.

How to read this report

Following this description, you will see the identifying information for your analysis. Under this information, we list the number of unique APIs/exports we received from you with the count of those we 'recognized' as system-related functions and the count of those we did not recognize. We generally find that the unrecognized ones are from your own application or the additional middleware that your application uses. If you see lists of APIs in the 'unrecognized' section of the report that you do not recognize as your

application APIs, you will need to identify the other products your application is using so we can verify support for those products too on AS/400.

After the counts, you will see two tables with several rows and right/left sections. Each row represents an AS/400 release. The two major right/left sections are analyses for the AS/400 Integrated Language Environment (ILE) and for the AS/400 Portable Application Solutions Environment (AS/400 PASE), respectively. The first table addresses the specific counts of APIs; the second addresses the percentages of APIs that fall into the categories for a quick overview analysis of the data.

In the first table, within each row you will see the count of APIs that are generally supported and unsupported for each environment. Each number in the table is a link to a following part of the report. If you click on the number, you will be taken to the list of the APIs that fall into that category, with some additional information as appropriate.

The 'flags' columns list additional considerations or workarounds for using a particular API for a particular release. Flags appear on both generally supported and unsupported APIs to point you to additional implementation details that could affect your application. Please read through these to better understand your port assessment. In particular, if you use system() or popen() functions, check your source code to see what commands or utilities you are running and check our Qshell Web page at `http://www.as400.ibm.com/developer/qshell/utils.html` or AS/400 PASE Web pages to ensure that support is available for those as well. Additional considerations for ILE and AS/400 PASE porting are located at `http://www.as400.ibm.com/developer/factory/table.html`

Again, thank you for your interest.

Company and Application Information

Company name	Test library
Product name	Daryl.report
Product API DB file name	DARYLTEST
Analysis report date	Wednesday, June 14, 2000 4:20 PM
Report prepared for OS/400 release	V4R5

Summaries

Total APIs for review:	1194
User defined or unrecognized by analysis:	591
APIs recognized by analysis:	603

API count

ILE					PASE				
VRM	APIs supported		APIs not supported		VRM	APIs supported		APIs not supported	
		With flags		With flags			With flags		With flags
V4R5	142	19	461	0	V4R5	602	17	1	0

Percentages

ILE					PASE				
VRM	APIs supported		APIs not supported		VRM	APIs supported		APIs not supported	
		With flags		With flags			With flags		With flags
V4R5	23%	19	77%	0	V4R5	99%	17	1%	0

User Defined or Unrecognized APIs (591)

__ct__8ifstreamFPCciT2
__ct__8ofstreamFPCciT2
__ct__9strstreamFv
__dbargs
__dbsubc
__dbsubg
__dbsubn
__dl__FPv
__dt__8ifstreamFv
__dt__8ofstreamFv
__dt__9strstreamFv
__frame_ptr
__ls__7ostreamFPCc
__my_pc_address

__nw__FUI
__vec__delete2
__vtt8ifstream
__vtt8ofstream
__vtt9strstream
__BadRethrow
__CleanupCatch
__CurrentException
__DestructSOMBase
__ExtractRegs
__PureVirtualCalled
__ResetStackPointer
__RestoreRegsAndJump
__SaveNonVolatileRegisters
cerr
close__11fstreambaseFv
complicated_put__7ostreamFc
endl__FR7ostream
gcount__7istreamFv
getline__7istreamFPcic
libhd_dl__FPv__FPviN22PFPcUI_vUI
libhd_nw__FUI__FUIiT2PFPcUI_v
m_DXLOutput
m_Pick
openprot__7filebuf
read__7istreamFPci
seekg__7istreamFl
seekg__7istreamFlQ2_3ios8seek_dir
terminate__Fv
...

[ILE information is next in the full report]

PASE V4R5 Supported List (602)

__assert
__divss
__divus
__filbuf
__flsbuf
__lc_charmap
__mulh
__quoss
__quous
__start
_iob
_system_configuration
_DBLINF
_Xm_fastPtr
_XmDifferentBackground
_XmDispatchGadgetInput
_XmDrawShadow
_XmDrawShadowType
_XmForegroundColorDefault

_XmGetClassExtensionPtr
_XmHighlightBorder
_XmInputInGadget
_XmManagerEnter
_XmManagerFocusIn
_XmMoveObject
_XmPrimitiveEnter
_XmPrimitiveLeave
_XmResizeObject
_XmStrings
_XmUnhighlightBorder
_XtAppInitialize
_XtCheckSubclassFlag
_XtInherit
_XtInheritTranslations
_XtInitialize
_XtIsSubclassOf
_XtToolkitInitialize
_XtVaAppInitialize
abort
abs
alarm
applicationShellWidgetClass
atexit
atof
atoi
bcmp
bind
bsearch
bzero
calloc
chdir
chmod
close
closedir
connect
creat
cs
ctime
dup2
environ
errno
execl
execv
execve
execvp
exit
fclose
fdopen
fflush
fgetc
fgets
fopen
fork

fp_raise_xcp
fp_set_flag
fprintf
fputc
fputs
fread
free
fscanf
fseek
fstat
ftell
fwrite
getcwd
getdtablesize
getenv
geteuid
gethostbyaddr
gethostbyname
gethostname
getlogin
getopt
getpid
getpwnam
getpwuid
getsockopt
gettimeofday
herror
inet_addr
ioctl
isalnum
isalpha
isascii
isatty
isdigit
isgraph
islower
isprint
ispunct
isspace
isupper
isxdigit
kill
listen
load
loadbind
loadquery
lseek
malloc
mbsinvalid
mbtowc
memcmp
memset
mkdir
mktemp

naccept
ngetsockname
open
opendir
optarg
overrideShellWidgetClass
pause
pclose
perror
pipe
popen
printf
putenv
qsort
random
re_comp
re_exec
read
readdir
realloc
rename
rewind
rint
sbrk
select
setbuf
setlocale
setsockopt
setvbuf
shmat
shmctl
shmget
sigaction
sigemptyset
signal
sigprocmask
socket
sprintf
srandom
sscanf
stat
strcasecmp
strchr
strcspn
strdup
strerror
strlen
strncmp
strrchr
strspn
strstr
strtod
strtol
strtoul

sys_errlist
sys_nerr
sysconf
system
time
times
tolower
topLevelShellWidgetClass
toupper
transientShellWidgetClass
umask
uname
ungetc
unlink
unload
usleep
vendorShellClassRec
vsprintf
wait
wait3
wcstombs
wctomb
widgetClassRec
write
...[long list of X Window and Motif APIs]
XWindowEvent

PASE V4R5 Supported Flagged (17)

API name	API type	Comment
_system_configuration	CONF	Subset for kernel/hardware differences
creat	FILES	No S_ISVTX or S_ENFMT support
getpwnam	AUTH	PASE always sets pw_gecos to zero
getpwuid	AUTH	PASE always sets pw_gecos to zero
getsockopt	SOCK	Partial support
ioctl	DEV	Partial support
isatty	TTY	PASE returns TRUE for the (non-tty) descriptors initially assigned to fd 0/1/2; environment variable PASE_STDIO_ISATTY=N overrides this behavior
lseek	FILES	IFS restricts directories to sequential access

API name	API type	Comment
open	FILES	No S_ISVTX S_ENFMT O_DEFER O_SYNC O_DSYNC FAIO support; O_DELAY and O_NOCTTY ignored
popen	PROC	Please examine source code to see what is called; check utilities lists for AS/400 as needed
setsockopt	SOCK	No support for SO_CKSUMRECV, IP_HDRINCL, IP_RECVOPTS, IP_RECVRETOPTS, IP_RECVDSTADDR, IP_RETOPTS, IP_RECVMACHDR, IP_RECVIFINFO_AIX, IP_BROADCAST_IF, IP_DHCPMODE
shmat	SHM	shmat of a file descriptor not supported
sysconf	PROC	Partial support
system	CLIB	Please examine source code to see what is called; check utilities lists for AS/400 as needed
time	TIME	PASE does not round up to next second (ILE does)
times	PROC	Subset support (only user time)
wait3	PROC	Supports RUSAGE_SELF and RUSAGE_CHILDREN only

PASE V4R5 Unsupported List (1)

API name	API type
absinterval	TIME

PASE V4R5 Unsupported Flagged (0)
None

IBM API analysis tool level: Version 0.9b
AS/400 API database level: March 9, 2000

Chapter 6. Porting mechanism

This chapter deals with some of the issues and means of porting a UNIX application to AS/400 PASE. Examples of such ports are described in Chapter 10, “AS/400 PASE porting examples” on page 147. Issues involved in an application port that uses a database are described in Chapter 7, “Database porting with AS/400 PASE” on page 87.

Whenever possible, porting projects should be debugged on an AIX system before being ported to AS/400 PASE. Applications that integrate into the AS/400 system through DB2/400 or ILE interfaces require different debugging strategies. See Chapter 12, “Problem determination and messages” on page 173, for more information.

6.1 Beginning a port

AS/400 PASE programs are structurally identical to AIX executables for PowerPC. They can be built using any AIX compiler and linker that generate output compatible with the AIX Application Binary Interface (ABI) for PowerPC and whose runtime is supported in AS/400 PASE. AS/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).

AS/400 PASE supports executables built for AIX 4.3.3 (and usually earlier 4.3 AIX releases).

AS/400 PASE implements a subset of AIX syscalls that allow many AIX programs (root executables and shared libraries) to run unchanged. The V4R5 version of AS/400 PASE runtime contains approximately 140 MB of executables.

AS/400 PASE defaults to delivering SIGILL (illegal instruction) to any program that tries to use a syscall that is not supported by AS/400 PASE. See 6.6.2, “AS/400 PASE unsupported system calls” on page 85, for information on how to change this behavior. The syscall that caused the error can be identified by looking at VLOG entries, using SLIC trace facilities (which you can activate using the TRCINT CL command for TRCTYPE(*PASE)), or running dbx against a core dump file.

6.2 Compiling applications on AIX for AS/400 PASE

There are two types of ports to consider at this point. One type of port is of an application that uses only functions exported by the AIX version of runtime libraries supplied for AS/400 PASE. You can simply compile and link with AIX libraries to prepare AS/400 PASE binaries. This is the case in the example application ports on GNU zip and GNU perl described in 10.1, “GNU zip” on page 147, and 10.3, “GNU perl” on page 159.

The second kind of port is one that intends to use functions that exist on OS/400, including native database access. These extensions to the AIX runtime support are defined in as400_protos.h. You need to provide definitions for the extensions when you link the application. You can do this either with an exports file that defines the AS/400-unique library exports, or by linking with AS/400 PASE runtime libraries (which must be accessible on the AIX system used to build the application, perhaps through NFS).

Attention

We recommend using the library export files, instead of linking directly with the AS/400 PASE runtime libraries, to reduce the risk of damaging the AIX system with the wrong libraries. Using the unique AS/400 PASE export files requires source code changes.

The available header and export files are shown in Appendix A, “Programming resources” on page 203.

Note that the sqlcli.h file is an AS/400 unique file that interfaces to DB2/400 and not DB2 on AIX. See Chapter 7, “Database porting with AS/400 PASE” on page 87, for more information on porting to the native AS/400 database.

The as400_libc.exp file defines all the exports from the AS/400 PASE version of libc.a that are not exported by the AIX versions of those libraries. The libdb400.exp file defines the exports from the AS/400 PASE libdb400.a library (DB2/400 CLI support).

If you have AS/400 PASE runtime libraries on your AIX systems, the easiest way to link with them is to use the -Z loader option to specify the name of a directory that contains the “lib” directory where AS/400 PASE runtime libraries are stored. The -Z option is supported on many compiler commands that call the linker (ld command).

If your program needs to use AS/400-unique functions in AS/400 PASE (such as `_ILECALL` to call an ILE procedure, or using the ILE pointer value type to store a tagged pointer), you need additional header files and build options.

AS/400 PASE augments standard AIX runtime with a few header files for AS/400-unique support. Header file as400_types.h declares the ILE pointer value type that can be used for 16-byte (quadword-aligned) tagged MI pointers. The xlc compiler provides limited support for 16-byte alignment (for type long double) by using the combination of `-qlngdbl128` and `-qalign=natural`. Use of type ILEpointer requires these compiler options to ensure that MI pointers are quadword 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.

Note

The xlc/xlC compiler currently does not provide a way to force quadword alignment for static or automatic variables. The compiler only guarantees relative alignment for 128-bit long double fields within structures. But the AS/400 PASE version of malloc always provides quadword-aligned storage, and you can arrange quadword alignment of stack storage. See 8.3, “Doing callouts to ILE from AS/400 PASE” on page 108, for details on how to provide quadword alignment for static and automatic variables.

Header file as400_types.h also relies on the long type 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). Assuming AS/400-unique headers and export files are in /home/spartz/pase/include, the following examples build the same program with the same options:

```
xlc    -o as400_test -qldbl128 -qlonglong -qalign=natural -H16
           -l c128
           -I /pase/exports
           -bI:/pase/include/as400_libc.exp
           as400_test.c
```

If the program is multi-threaded, use the `xlc_r` command to ensure the executable links with threadsafe runtime libraries:

```
xlc_r  -o as400_test -qldbl128 -qlonglong -qalign=natural -H16
          -l c128
          -I /pase/exports
          -bI:/pase/include/as400_libc_r.exp
          as400_test.c
```

Note

If you are using the AS/400 PASE DB2 UDB CLI support, you also need to specify `-bI:/pase/include/libdb400.exp` on your build command.

-H16 is a loader option to force 16-byte alignment for each section of the output file. Note that the xlc compiler only requests 8-byte alignment for any allocation within BSS, so -H16 does not currently have the desired effect. We recommend using -H16 so the entire BSS area will be quadword aligned.

The `-bI` directive tells the compile to pass the parameter to the linking directive ID, `ld`. The directive specifies an export file containing exported symbols from a library, to be imported by the linking directive ID for symbol resolution.

6.3 Accessing files from AS/400 PASE

There are several techniques for referencing or copying files from the AIX development platform to AS/400 PASE. AS/400 PASE uses the AS/400 IFS file system, so all methods of accessing AS/400 IFS can achieve the desired result.

6.3.1 Using FTP

AS/400 PASE does not provide, nor does it need to supply, an FTP daemon or client. The native OS/400 FTP daemon and client can be used to transfer a file into, or out of, the OS/400 IFS file system. Non-text files should be transferred in binary mode. Use the `bin` FTP subcommand to set this mode. Not all FTP clients default to binary. To switch back to text mode, use the `ascii` or `ebcdic` FTP subcommand, depending on the source encoding type.

Placing files into the IFS requires the use of naming format 1 of the AS/400 FTP server. This format allows the use of UNIX path names. 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. This means ensuring that the first directory is prefaced by a "/", 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.

Note

The FTP standard calls for data sent to use carriage return/line feed (CRLF) for line end in text mode. 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 AS/400 PASE runtime 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:

```
find . -type f -print | sed 's/^\(.*\)$/cat "\1" | cat - | tr -d \\\\\r >
"\1"/' | ( sh )
```

6.3.1.1 Using the OS/400 FTP daemon

Data or files are transferred using the `put` FTP subcommand. Use the `lcd` FTP subcommand to change the local directory (your workstation) for the source objects. Use the FTP subcommand `cd` to set the remote (AS/400 IFS) directory. Name format 1, as stated above, is required to store files into the stream file part of the IFS name space. On the client FTP session, enter `quote site namefmt 1`. Remember to set the transfer mode (binary, ascii, or ebcdic) as necessary before using `put`.

6.3.1.2 Using the OS/400 FTP client

Data or files are transferred using the `get` FTP subcommand. Use the `lcd` FTP subcommand to change the local (AS/400 system) directory to the destination. If needed, namefmt is supported by the AS/400 FTP client program. Enter `quote site namefmt 1`. Use the `cd` FTP subcommand to set the remote directory. Remember to set the transfer mode (binary, ascii, or ebcdic) as necessary before using `get`.

6.3.2 Using SMB

The AS/400 system supports SMB client and server components. With NetServer configured and running, the AS/400 PASE environment 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 AS/400 PASE.

6.3.3 Remote file systems

OS/400 includes the ability to mount NFS file systems to a mount point in the IFS file space. AIX supports NFS, as well as DFS and AFS, 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, allows AS/400 PASE applications to 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.

6.4 Configuration tips

Some important tips that can help during configuration include:

- Some shell scripts may specify the shell from the /bin directory. For example, a shell may look like the following example:

  ```
  #!/bin/sh
  ```

  The /bin directory, by default, does not exist. All shell scripts that rely on this can be changed to:

  ```
  #!/QOpenSys/usr/bin/sh
  ```

  Or, a symbolic link can be generated for /bin to point to /QOpenSys/usr/bin:

  ```
  ln -s /QOpenSys/usr/bin /bin
  ```

- Programs or scripts that use hardcoded paths to utilities in /usr/bin will have problems. The /usr/bin path is used by the AS/400 system for the QSHELL environment, an EBCDIC shell, and that directory contains OS/400 programs, which cannot be run within AS/400 PASE. The AS/400 PASE equivalent directory is /QOpenSys/usr/bin. If other file systems are not required, an alternative is to use the AS/400 PASE utility `chroot` to change the base of the root file system, for example:

  ```
  mkdir -p /QOpenSys/tmp
  chroot /QOpenSys /QOpenSys/usr/bin/sh -i
  ```

This creates a temporary directory for sh (trying the `chroot` utility without the tmp directory will fail) and establishes /QOpenSys as / for the new interactive instance, sh -i (see Figure 5 for an example of this process).

```
                              /QOpenSys/usr/bin/sh

  QCA400         QNTC           asakai         itscid61       schimu
  QDLS           QNetWare       bin            ldapdir        tmp
  QFPNWSSTG      QOPT           cisco          netsecrb       usr
  QFileSvr.400   QOpenSys       core           notes          var
  QIBM           QSR            dev            ntins          xclock
  $
> mkdir -p /QOpenSys/tmp
  $
> chroot /QOpenSys /QOpenSys/usr/bin/sh -i
  $
> ls /
  QIBM      QSR       core      home      spss      usr
  QOpenSys  bin       etc       schimu    tmp       var
  $

===>



F3=Exit   F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
F13=Clear F17=Top  F18=Bottom  F21=CL command entry
```

Figure 5. Mapping to /QOpenSys/usr/bin

- Local installed tools are typically put in /usr/local/bin. It may be useful to set the environment variable PATH in the shell you are running to include this. For example, in Bourne and Korn shell:

  ```
  export PATH=/usr/local/bin:$PATH
  ```

  In C shell:

  ```
  setenv PATH /usr/local/bin:$PATH
  ```

- For X client applications, the environment variable DISPLAY must be set to the X server that will render the graphics. The X server runs on the user's workstation. The variable can be set to either the hostname, hostname:0 (if the hostname is resolvable through DNS), or the users IP address, IP_address:0. For example, in Bourne and Korn shell:

  ```
  export DISPLAY=itscid49:0
  ```

  In C shell:

  ```
  setenv DISPLAY=itscid49:0
  ```

  You also need to make sure the X server system is authorized to the AS/400 system for host access. There is not a common way of doing this

across different packages, but it can be a common problem for not being able to connect.

- A user's home directory for AS/400 PASE and OS/400 QSHELL is selectable within the OS/400 user profile. Use the `CHGUSRPRF` command to change the field HOMEDIR to the desired home directory. Ensure that the home directory is created, since it may not have been created when the user profile was created.
- Job environment variables are passed to the AS/400 PASE environment only when it is started via the launchers. Variables prefixed with PASE_ are modified within the startup of the AS/400 PASE environment by having the prefix stripped. Use the `WRKENVVAR` command to change, add, or delete environment variables as needed prior to starting AS/400 PASE.
- There are several places in which case sensitivity may cause complications with existing code, for example Qp2Shell and Qp2Term. UNIX-like environments differentiate between uppercase and lowercase letters, where on OS/400 that is not always the case. Case sensitivity on a directory or file basis will only work correctly from the /QOpenSys file system. Other areas to be aware of are user IDS and group IDS on OS/400. Some notable places where case (in)sensitivity may be a problem are:

  a.
  ```
  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
  ```
  The shell does a character comparison of the generic name prefix against what is returned by `readdir()`. However, the file system is case sensitive, and therefore, no results are returned in the first instance.

  b.
  ```
  find /qsys.lib/v4r5m0.lib/ -name 'qwobj*' -print

  find /qsys.lib/v4r5m0.lib/ -name 'qwobj*' -print
  /qsys.lib/v4r5m0.lib/QWOBJ.FILE
  ```
  This is similar to the first case (a) except that, in this case, the find utility is doing the comparison and not the shell.

  c.
  ```
  ps -utimms -f
  UID PID PPID C STIME TTY TIME CMD

  ps -uTIMMS -f
  UID PID PPID C STIME TTY TIME CMD
  ```

```
TIMMS 279 275 o 11:00:05 - 3477205:31 /QOpenSys/usr/bin/sh -i
TIMMS 321 279 0 14:55:53 - 3477205:31 ps -uTIMMS -f
```

The `ps` utility compares the string specified for the -u flag with what comes back from `getprocs()`, but it does not know that user names are case insensitive.

6.5 Starting an AS/400 PASE application

There are several techniques for starting an application in the AS/400 PASE environment, from a terminal session, or programmatically from an application.

Two methods, QP2SHELL and QP2TERM, can be used to start AS/400 PASE programs from the OS/400 command line. These are two ILE programs that are directly called.

Two APIs, QP2RunPase and Qp2CallPase, are used within ILE programs to access AS/400 PASE programs. QP2RunPase is used to initialize the AS/400 PASE environment and cannot be used recursively. QP2CallPase is used after the AS/400 PASE program has done an ILE callout when the program logic requires another AS/400 PASE call. AS/400 PASE calls can be made back into the same AS/400 PASE instantiation in a job using this method.

6.5.1 Invocation from a 5250 terminal screen: QP2SHELL

An application can be started in the AS/400 PASE environment by using the QP2SHELL callable MI command and passing the application name to launch as the parameter, for example:

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

`QP2SHELL()` runs an AS/400 PASE program in the job where it is invoked. An example of the output is shown in Figure 6 on page 66. QP2SHELL is further explained in the following sections.

```
 CALAB          QISAFIX        QSYS.LIB       home           pfizer
MGTCTXXDIR     QJAVA          QTCPTMM        itscid61       schimu
QCA400         QNTC           asakai         ldapdir        tmp
QDLS           QNetWare       bin            netsecrb       usr
QFPNWSSTG      QOPT           core           notes          var
QFileSvr.400   QOpenSys       dev            ntins          xclock
QIBM           QSR            etc            paymentserver
Press ENTER to end terminal session.









===>

F3=Exit F4=End of File F6=Print F9=Retrieve F17=Top
F18=Bottom  F19=Left   F20=Right F21=User Window
```

Figure 6. Sample output from QP2SHELL CL command

6.5.1.1 Syntax

```
#include <qp2shell.h>
QP2SHELL     (const char         *pathName,
              [const char         *argString ... ] )
```

6.5.1.2 Parameters

The first argument for QP2SHELL is the path name for an AS/400 PASE program. Additional arguments are optional. Copies of all argument strings are passed to the AS/400 PASE program (following the C language convention that the first argument is the name of the program itself).

Note

When invoking QP2SHELL from CL, be sure to quote any argument string that could be interpreted as a numeric value. CL converts unquoted numeric arguments to the decimal or floating-point format, which does not match the assumption made by QP2SHELL and AS/400 PASE programs that all arguments are null-terminated character strings.

QP2SHELL initializes AS/400 PASE environment variables with a modified copy of the entire ILE environment. An AS/400 PASE environment variable is

initialized for every ILE environment variable, but the initial value of any AS/400 PASE variable (except those with a name beginning “PASE_”) is overridden by the value of an environment variable with a name that concatenates the prefix PASE_ with the original variable name. This process avoids some interference between AS/400 PASE runtime and ILE runtime when they require different values for the same environment variable (for example, LANG).

QP2SHELL initializes any of these ILE environment variables that are not already set to provide default values used by AS/400 PASE programs:

PASE_PATH = "/QOpenSys/usr/bin:/usr/ccs/bin:/usr/sbin:.:/usr/bin"
Specifies the initial value for the AS/400 PASE PATH environment variable.

PASE_LANG and QIBM_PASE_CCSID
Specifies the initial value for the PASE LANG environment variable and tells the system what coded character set identifier (CCSID) the AS/400 PASE program will use. QP2SHELL sets both these environment variables if either or both are absent. The default values are a function of the current LANGID and CNTRYID attributes of the job. However, the system will use PASE_LANG=POSIX and QIBM_PASE_CCSID=819 if it does not recognize the LANGID and CNTRYID pair.

PASE_LOCPATH = “/usr/lib/nls/loc”
Specifies the initial value for the PASE LOCPATH environment variable.

PASE_NLSPATH = “/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat”
Specifies the initial value for the PASE NLSPATH environment variable.

PASE_LC__FASTMSG = “true”
Specifies the initial value for the PASE LC__FASTMSG environment variable.

PASE_TZ = “”
Specifies the initial value for the PASE TZ environment variable. If no time zone information is provided in environment variable TZ, the AS/400 PASE program sees Coordinated Universal Time (UTC) as local time.

LOGIN If LOGIN is not set, QP2SHELL sets it to the middle qualifier of the job name. For an interactive job, this is the name of the user who did a signon to start the job.

HOME If HOME is not set, QP2SHELL sets it to the home directory path specified in the user profile identified by the LOGIN variable. If the job is not currently authorized to the LOGIN user profile, the HOME environment variable is set to a null string.

QIBM_IFS_OPEN_MAX = "33000"
Specifies the maximum number of IFS open file descriptors desired in the job. QP2SHELL attempts to set the maximum number of IFS descriptors using this environment variable and updates the variable to reflect the actual limit (in case the requested limit is not currently allowed). Any change to the maximum number of file descriptors persists after QP2SHELL returns.

AS/400 PASE programs assume the ability to open 32,767 files, and the system requires an open file for each AS/400 PASE executable it loads. Therefore, the default of 33,000 files accommodates a maximally large AS/400 PASE program with a fairly large number of loaded executables.

6.5.1.3 Programming notes

QP2SHELL uses the Qp2RunPase API to run the AS/400 PASE program.

QP2SHELL sets the ILE pthread cancel state and cancel type to default values (PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DEFERRED) prior to running the AS/400 PASE program. This is done to avoid unexpected behavior for the AS/400 PASE program if the job changed ILE pthread attributes prior to calling QP2SHELL.

QP2SHELL sets up handlers for all ILE signals while the AS/400 PASE program runs. The handlers call the Qp2SignalPase() API to post an equivalent AS/400 PASE signal to the AS/400 PASE program. QP2SHELL restores the original ILE signal handling before returning to the caller.

To avoid unpredictable results, do not change ILE environment variables `QIBM_USE_DESCRIPTOR_STDIO` or `QIBM_PASE_DESCRIPTOR_STDIO` in a job where an AS/400 PASE program is running. Also avoid running ILE code in threads that are not created by the AS/400 PASE program.

You may want to set the ILE environment variable PASE_TZ at the system level to provide a default time zone other than UTC for AS/400 PASE programs. For example, this CL command sets the default time zone to US Central time:

```
ADDENVVAR ENVVAR(PASE_TZ) VALUE('CST6CDT') LEVEL(*SYS)
```

To run several commands in the AS/400 PASE environment through QP2SHELL, you can type:

```
CALL PGM(QP2SHELL) PARM('/QOpenSys/usr/bin/ksh' 'cd
/home/bhenry/test;env;pwd;echo "Hello World"')
```

In this case, you are passing one command to the QP2SHELL, '/QOpenSys/usr/bin/ksh', but everything that follows is considered a parameter to that initial command, as long as the parameter string is enclosed in quotes. The semi-colons are standard list delimiters for UNIX command lines since this parameter string is passed (untouched) to the ksh.

Some commands are output to the screen, while others are sent to the spooled file (WRKSPLF). In the previous example, the output from the **env** command is sent to the spooled file, while the output from the **pwd** and **echo** commands are sent to the screen. A successful **cd** command doesn't generate any output. But, unsuccessful **cd** commands send their error messages to the screen. If you don't see the expected output on the screen, check the latest QPRINT (F18 takes you to the bottom where the latest entries are) entry in the WRKSPLF display.

6.5.1.4 Error conditions

QP2SHELL re-signals any exception message sent by the Qp2RunPase API. It also sends escape messages if it detects errors prior to attempting to run the AS/400 PASE program.

6.5.2 Invocation from the AS/400 PASE terminal: QP2TERM

The AS/400 PASE terminal environment is started by the command:

```
call qp2term
```

The default shell prompt is written to the screen once the terminal session is ready. Any of the utilities listed in Table 3 on page 15 can be invoked with `stdout` and `stderr` written and scrolled in the terminal screen. Any PASE program can also be invoked from here. Figure 7 on page 70 shows a terminal session. The `pwd` commands and `ls` were executed and the output was displayed in the terminal session.

Program QP2TERM() runs an “interactive terminal session” that submits a batch job to run an AS/400 PASE program in a batch job. It uses the workstation display in the interactive job to present output and accept input for files `stdin`, `stdout`, and `stderr` in the batch job. QP2TERM is further explained in the following section.

```
                        /QOpenSys/usr/bin/sh

  $
> pwd
  /home/SPARTZ
  $
> ls
  id.pl   opendx  perl
  $









===>


F3=Exit   F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
F13=Clear F17=Top  F18=Bottom  F21=CL command entry
```

Figure 7. Sample output from QP2TERM terminal session

6.5.2.1 Syntax

```
#include <qp2term.h>
QP2TERM [ (    const char         *pathName,
               const char         *argString ... ) ]
```

6.5.2.2 Parameters

Any argument strings for QP2TERM are passed to program QP2SHELL in the batch job. The arguments specify a path name and argument data for an AS/400 PASE program that runs in the batch job. If QP2TERM is invoked without arguments, it runs the default AS/400 PASE shell, /QOpenSys/usr/bin/sh, with a single argument of “-i” (to run an interactive shell).

Note

When invoking QP2TERM from CL, be sure to quote any argument string that could be interpreted as a numeric value. CL converts unquoted numeric arguments to decimal or floating-point format, which does not match the assumption made by QP2TERM and AS/400 PASE programs that all arguments are null-terminated character strings.

The environment for the batch job is initialized by copying environment variables from the interactive job and adding or replacing some variables. These changes only affect the batch job, they do not modify the environment in the job that called QP2TERM:

ROWS If ROWS is not set in the interactive job, it is initialized in the batch job to the number of rows available for program output on one line of the interactive display presented by the terminal manager session.

COLUMNS If COLUMNS is not set in the interactive job, it is initialized in the batch job to the number of columns available for program output to one of the interactive displays presented by the terminal manager session.

QIBM_USE_DESCRIPTOR_STDIO=I
This environment variable is set (unconditionally) in the batch job to ensure that `stdin`, `stdout`, and `stderr` use file descriptors 0, 1, and 2, which are associated with pipes the terminal manager uses to communicate with the batch job.

QIBM_PASE_DESCRIPTOR_STDIO=T
This environment variable is set (unconditionally) in the batch job to ensure that AS/400 PASE runtime does ASCII/EBCDIC text conversion for data that the AS/400 PASE program reads or writes to files `stdin`, `stdout`, and `stderr`. The binary value B may not be used.

6.5.2.3 Programming notes

The batch job for QP2TERM is submitted by the Qp0zStartTerminal API, which copies most of the attributes of the interactive job. For example, you can force the batch job to produce a job log by changing the interactive job (using the CHGJOB command) to LOG(4 0 *SECLVL) before invoking QP2TERM.

The batch job calls program QP2SHELL to run the AS/400 PASE program. See QP2SHELL documentation for information about controlling various options that apply to running an AS/400 PASE program.

6.5.3 Invocation from an ILE application: Qp2RunPase

The Qp2RunPase API (to run an AS/400 PASE program) allows the caller to initiate AS/400 PASE use, specifying the program name, argument strings, and environment variables, similar to the industry-standard `execve()` interface. In addition, the API requires the user to specify the (ASCII) CCSID for the AS/400 PASE program (an initial value that can be modified by the

AS/400 PASE program using the _SETCCSID interface). It also allows arbitrary data (including tagged pointers) to be copied into memory that the AS/400 PASE program can reference through an external symbol.

The Qp2RunPase() function runs an AS/400 PASE program in the job where the API is invoked. It loads an AS/400 PASE program (including any necessary shared libraries) and then transfers control to the program. It is only used to initiate the AS/400 PASE environment and returns an error if AS/400 PASE is already active.

Qp2RunPase() includes the ability to run shell scripts and the rules for resolving shared libraries (sometimes using the LIBPATH environment variable). Qp2RunPase does not overlay or destroy the state of any ILE code currently active in the process, so control returns to the caller of Qp2RunPase when the AS/400 PASE program runs the exit function.

The Qp2CallPase API (to call a function in an AS/400 PASE program) provides support for ILE to call back into an AS/400 PASE instantiation that has already been started.

Applications in AS/400 PASE run unprivileged, so they must use the PowerPC System Call instruction (assembler mnemonic sc) to access any service outside of AS/400 PASE libraries or to return (exit) from AS/400 PASE. SLIC services all AS/400 PASE system calls and either provides the service itself or transfers control to an ILE procedure in the SLS environment that provides the service. SLIC (in combination with processor hardware support) automatically sets the processor state to tags-active mode when fielding a system call from AS/400 PASE. It also restores tags-inactive mode when returning to AS/400 PASE.

Note

Qp2RunPase calls an AS/400 PASE program and not procedures or functions. Therefore, the code you are calling must have a main() entry point.

6.5.3.1 Syntax

```
#include <qp2user.h>
int
Qp2RunPase(const char           *pathName,
           const char           *symbolName,
           const void           *symbolData,
           unsigned int         symbolDataLen,
```

```
int                 ccsid,
const char          *const *argv,
const char          *const *envp);
```

The service program is QP2USER.

6.5.3.2 Authorities and locks

The user must have read and execute authority for the stream file that contains the AS/400 PASE program (identified by the pathName argument), and read authority to every stream file containing a shared library required by the program.

The system opens the stream files containing the program and its dependent shared libraries for input. It leaves the files open until the AS/400 PASE program terminates.

6.5.3.3 Parameters

pathName Input; Pointer to Char(*)

pathName is a null-terminated character string that identifies the stream file in the integrated file system (IFS) that contains the AS/400 PASE program to run. The pathName string may include an absolute or relative path qualifier in addition to the stream file name. Relative path names are resolved using the current directory for the job (maintained by the IFS).

symbolName Input; Pointer to Char(*)

symbolName is a null-terminated character string that specifies the name of an external symbol in the AS/400 PASE program that should be initialized with the data specified by the symbolData and symbolDataLen arguments. The symbol must be defined (not just referenced) in the AS/400 PASE program, although the AS/400 PASE loader relocates it to memory that is dynamically allocated (to ensure 16-byte alignment).

The system copies the symbolName string internally and converts it from the job default CCSID to the CCSID specified by the ccsid argument before searching for the (converted) symbol name in the AS/400 PASE program.

The AS/400 PASE program is run without initializing symbol data (and no errors are reported) if either the symbolName argument is

a null pointer or the specified symbol name is not an external (imported or exported) symbol in the AS/400 PASE program.

symbolData Input; Pointer to any data

symbolData specifies the address of data used to initialize a symbol (identified by the symbolName argument) in the AS/400 PASE program. The symbolData argument is ignored if the symbolName argument is a null pointer.

The system copies the symbolData (without modification) into memory that can be referenced by the AS/400 PASE program. Any (complete) 16-byte MI pointer values in the symbolData are preserved in the copy presented to the AS/400 PASE program. Tagged MI pointers in AS/400 PASE memory are destroyed by exec processing and are also unusable in a fork child process. It may be better to have the AS/400 PASE program use the _ILELOAD and _ILESYM syscalls to acquire tagged pointers to functions and data exported by an ILE program or service program rather than rely on pointers passed using the symbolName, symbolData, and symbolDataLen arguments on Qp2RunPase.

symbolDataLen
Input; Unsigned Binary(4)

symbolDataLen specifies the length of data (located by the symbolData argument) to initialize in the AS/400 PASE program for the symbol identified by the symbolName argument. The symbolDataLen argument is ignored if the symbolName argument pointer is null.

ccsid Input; Binary(4)

ccsid specifies the Coded Character Set IDentifier (CCSID) initially used by the AS/400 PASE program. The ccsid value must specify a single-byte encoding (normally an ASCII CCSID) that OS/400 can convert to and from the job default CCSID, or a value of 1208 to indicate that the AS/400 PASE program uses UTF-8 encoding.

The system uses the ccsid value to set the CCSID of any IFS bytestream file created by the AS/400 PASE program, and also to control character encoding conversions done for AS/400 PASE runtime interfaces that use OS/400 services. The AS/400 PASE

program can use the _SETCCSID runtime interface to dynamically change the CCSID used for AS/400 PASE runtime interfaces or rebind to a change in the job default CCSID.

argv Input; Pointer to Array of Pointers to Char(*)

argv specifies the address of an array of space pointers that locate argument strings for the AS/400 PASE program. The arguments passed to an AS/400 PASE program are null-terminated character strings. The number of arguments passed is determined by the location of the first null pointer in the array of space pointers.

The system copies argument strings into AS/400 PASE memory and converts them from the job default CCSID to the CCSID specified by the ccsid argument. The standard entry point for an AS/400 PASE program passes the argument strings to the C main function. By convention, the first argument string passed to an AS/400 PASE program should be the same string specified for the pathName argument. At least one argument string should be passed (even if the AS/400 PASE program requires no arguments) to avoid failures in system code and utilities that assume the first argument is the program name.

envp Input; Pointer to Array of Pointers to Char(*)

envp specifies the address of an array of space pointers that locate environment strings for the AS/400 PASE program. Environment variables for an AS/400 PASE program are null-terminated character strings. The number of environment variables initialized for the AS/400 PASE program is determined by the location of the first null pointer in the array of space pointers. envp can be a null pointer if the AS/400 PASE program requires no environment variables.

The system copies environment variable strings into AS/400 PASE memory and converts them from the job default CCSID to the CCSID specified by the ccsid argument. The copied or converted environment strings can be accessed by an AS/400 PASE program using standard C runtime interfaces.

Qp2RunPase checks these two environment variables to determine how to setup AS/400 PASE files `stdin`, `stdout`, and `stderr`:

- **QIBM_USE_DESCRIPTOR_STDIO**: When this environment variable is set to Y or I, both AS/400 PASE runtime and ILE C runtime use IFS file descriptors 0, 1, and 2 for `stdin`, `stdout`, and `stderr`. Otherwise, AS/400 PASE files `stdin`, `stdout`, and `stderr` are mapped to ILE C runtime files `stdin`, `stdout`, and `stderr` (and do not use IFS file descriptors).

  AS/400 PASE and ILE generally use different descriptor numbers for the same open file. However, when QIBM_USE_DESCRIPTOR_STDIO is set to Y or I, any operation against AS/400 PASE file descriptors 0, 1, or 2 is also done for the same IFS file descriptor number. This way, AS/400 PASE and ILE C always use the same files for `stdin`, `stdout`, and `stderr`.

- **QIBM_PASE_DESCRIPTOR_STDIO**: This environment variable controls ASCII/EBCDIC conversion for data read or written through AS/400 PASE files `stdin`, `stdout`, and `stderr` to IFS file descriptors 0, 1, and 2. ASCII/EBCDIC conversion is always done (and this variable is ignored) unless QIBM_USE_DESCRIPTOR_STDIO is set to either Y or I. When QIBM_PASE_DESCRIPTOR_STDIO is set to B, the AS/400 PASE program processes binary data (without ASCII/EBCDIC conversion). Otherwise, ASCI/EBCDIC conversion is done for any data read from or written to AS/400 PASE file descriptors 0, 1, or 2.

6.5.3.4 Programming notes

Users of Qp2RunPase may want to invoke the DosSetRelMaxFH API to ensure the job has enough available file descriptors to run the AS/400 PASE program.

AS/400 PASE users should also consider establishing Qp2SignalPase as the handler for any (asynchronous) ILE signal that needs to be visible to the AS/400 PASE program. For example, system support for IFS and sockets used by AS/400 PASE only sends SIGIO and SIGURG as ILE signals, so ILE signal handling must be setup before invoking an AS/400 PASE program that relies on SIGIO or SIGURG as AS/400 PASE signals. AS/400 PASE runtime automatically establishes Qp2SignalPase as the handler for every ILE signal in a fork child job.

AS/400 PASE pthreads use ILE pthreads. Qp2RunPase assumes it is either invoked in a job where ILE pthreads are not yet initialized, or that the ILE pthread cancel state and cancel type are set to the defaults

(PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DEFERRED). The state of these ILE pthread attributes when an AS/400 PASE program ends is the value that was last set by either ILE or AS/400 PASE code. Users of Qp2RunPase may want to invoke ILE interfaces pthread_setcancelstate and pthread_setcanceltype to set ILE pthread cancel state and cancel type before running a multi-threaded AS/400 PASE program in a process that did prior pthread work (either ILE or AS/400 PASE).

Character conversions controlled by the ccsid argument only handle the single-byte component of an EBCDIC mixed CCSID (for the job default CCSID). This restricts argument strings passed through the argv argument to single-byte characters. Double-byte characters can be passed (unconverted) to an AS/400 PASE program using the symbolData argument.

AS/400 PASE runtime for sockets uses OS/400 sockets support, which restricts metadata arguments (such as Internet domain names) to single-byte characters. An AS/400 PASE program can use the entire UCS-2 character set for metadata arguments on file system interfaces (like file names) by setting the ccsid value to 1208 (UTF-8 encoding).

Any credentials changes (user, group, or group list changes) for an AS/400 PASE program are persistent in the calling job. The credentials used before and after a call to Qp2RunPase may not be the same if the stream file containing the AS/400 PASE program (or any AS/400 PASE program invoked by the exec interface) has the S_ISUID or S_ISGID attribute, or if the AS/400 PASE program invokes any of the setuid or setgid family of interfaces.

6.5.3.5 Possible errors returned

The following error messages can appear after invoking an AS/400 PASE program with QP2RunPase:

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

CPFB9C0 E Error loading program &1. See previous messages.

CPFB9C1 E System support for AS/400 PASE not available.

CPFB9C2 E Hardware support for AS/400 PASE not available.

CPFB9C3 E AS/400 PASE CCSID and job default CCSID are incompatible.

CPFB9FF E Internal error. Error code is &1.

6.5.3.6 Sample programs

The following sample shows a simple ILE C program to invoke an AS/400 PASE program and a simple AS/400 PASE program called from ILE.

Sample ILE program

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

/* include file for QP2RunPase. This header file
   must be obtained from IBM in V4R4 */
#include <qp2user.h>

/*****************************************
 Sample:
  A simple ILE C Program to invoke an AS/400
  PASE Program using QP2RunPase and
  passing one string parameter.
  When compiling this program in V4R4, bind
  to SRVPGM QP2USER.
  Example compilation:
   CRTCMOD MODULE(MYLIB/SAMPLEILE) SRCFILE(MYLIB/QCSRC)
   CRTPGM PGM(MYLIB/SAMPLEILE) BNDSRVPGM(QSYS/QP2USER)
*****************************************/
void main(intargc,     char*argv[])
 {
   /* Path name of PASE program */
   char *PasePath = "/home/samplePASE";
   /* Return code from QP2RunPase */
   int rc;
   /* The parameter to be passed to the
      AS/400 PASE program */
   char *PASE_parm = "My Parm\0";
   /* Argument list for AS/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 AS/400 PASE program. */
    rc = Qp2RunPase(PasePath,   /* Path name */
        (char *)NULL,   /* Symbol for calling to ILE, not used in this sample */
        (void *)NULL,   /* Symbol data for ILE call, not used here */
        0,              /* Symbol data length for ILE call, not used here */
        819,            /* ASCII CCSID for AS/400 PASE, always 819 in V4R4 */
        arg_list,       /* Arguments for AS/400 PASE program */
        (char **)NULL); /* Environment variable list, not used in this sample */

 }
```

Sample PASE program

```
#include <stdio.h>

/*****************************************
 Sample:
  A simple AS/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 AS/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 paramter passed in. Note arv[0] is the program
       name, so, argv[1] is the parameter  */
    printf("Hello I am in the AS/400 PASE program, parameter value is: \n %s \n",
argv[1]);

    return 0;

}
```

6.5.4 Invocation from an ILE application: Qp2CallPase

The Qp2CallPase() function calls a procedure in an AS/400 PASE program in the job where the API is invoked. Qp2CallPase() is only used to call into PASE in a job that already has the AS/400 PASE environment started, that is, if you had called ILE from AS/400 PASE and needed to make a call back to AS/400 PASE.

6.5.4.1 Syntax

```
#include <qp2user.h>
int
Qp2CallPase(   const void           *target,
               const void           *arglist,
               const QP2_arg_type_t *signature,
               QP2_result_type_t     result_type,
               void                 *result);
```

Service Program: QP2USER

6.5.4.2 Authorities and locks

No special authorization or locks are required.

6.5.4.3 Parameters

target Input; Pointer to an AS/400 PASE function descriptor

target is a pointer to a function descriptor for the procedure (in the AS/400 PASE program) to call. The format and contents of a function descriptor are specified by the PowerPC Application Binary Interface (ABI) for AIX. A function descriptor contains three AS/400 PASE addresses (not MI pointers) that point to the executable instructions, table of contents (TOC), and environment for the target procedure.

arglist Input; Pointer to AS/400 PASE argument list

arglist is a pointer to the argument list for the AS/400 PASE procedure. The format and contents of an AS/400 PASE argument list are generally specified by the PowerPC ABI for AIX. The specific argument list structure for the AS/400 PASE procedure identified by the target parameter is determined by the list of argument data types specified by the signature parameter.

signature Input; Pointer to array of Binary(2)

signature specifies the address of values that identify the data types of arguments passed to the AS/400 PASE procedure. The signature list must account for every byte of the arglist structure. It may require entries for bytes that are not actual argument data but are skipped for alignment. Header file qp2user.h defines the following constants for the data types supported as arguments to an AS/400 PASE procedure:

- **QP2_ARG_END**: This special value indicates the end of the list of argument type values.
- **QP2_ARG_WORD**: The argument is a 4-byte value that is either a signed or unsigned integer, or an aggregate no longer than four bytes. Doubleword and long aggregate argument types are passed as successive words under the rules of the 32-bit PowerPC ABI for AIX.
- **QP2_ARG_FLOAT32**: The argument is a 4-byte floating point value.
- **QP2_ARG_FLOAT64**: The argument is an 8-byte floating point value.

result_type

Input; Binary(2)

result_type specifies the data type of the function result returned

by the AS/400 PASE procedure. Header file qp2user.h defines the following constants for AS/400 PASE function result data types:

- `QP2_RESULT_VOID`: No function result is returned.
- `QP2_RESULT_WORD`: The function result is a 4-byte value that is either a signed or unsigned integer, or an aggregate no longer than four bytes.
- `QP2_RESULT_DWORD`: The function result is an 8-byte value that is either a signed or unsigned integer.
- `QP2_RESULT_FLOAT64`: The function result is an 8-byte floating point value.

`result` Output; Pointer to a buffer for the function result

result specifies the address of a buffer that will be updated with the function result (with a data type specified by the result_type parameter).

6.5.4.4 Restrictions

Qp2CallPase is only supported when an AS/400 PASE program is currently running in the job. This means that there must be an active invocation of the Qp2RunPase API in the job, or the job must be a fork child process.

An AS/400 PASE function invoked by Qp2CallPase must return to its caller. Unpredictable results occur if the AS/400 PASE function attempts to longjmp to an older invocation or if it performs an operation that terminates the thread or process (such as calling the exit function). If a signal handler is on the AS/400 PASE stack when Qp2CallPase is invoked, the called AS/400 PASE function must also honor restrictions on runtime functions allowed in signal handlers (see AIX signal handling documentation for details).

6.5.4.5 Programming notes

The address of any AS/400 PASE function is really the address of a function descriptor (required for the target argument on Qp2CallPase). This way, an AS/400 PASE program can easily provide a function descriptor to ILE code by passing a PASE function pointer value converted to an ILE memory address. The conversion can be done using the _SETSPP function or the ARG_MEMPTR argument type on the _ILECALL function.

6.5.4.6 Return value

The function result from the Qp2CallPase API indicates whether the AS/400 PASE function was called successfully. Header file qp2user.h defines the following constants for the return code from this API:

QP2CALLPASE_NORMAL

The AS/400 PASE procedure ran to completion and its function result (if any) was stored in the location identified by the result parameter.

QP2CALLPASE_RESULT_ERROR

The AS/400 PASE procedure ran to completion, but its function result could not be stored at the location identified by the result parameter. The result argument may be a null pointer value, or the space addressed by the operand could be damaged or destroyed.

QP2CALLPASE_ENVIRON_ERROR

The requested function is not allowed at this time because no AS/400 PASE program is currently loaded. You can only run the Qp2CallPase API while there is an active invocation of the Qp2RunPase API in the job.

QP2CALLPASE_ARG_ERROR

One or more argument values passed to the Qp2CallPase API are invalid.

QP2CALLPASE_TERMINATING

The AS/400 PASE program is terminating, so no function result was returned. The program may have run the exit function, or a signal might have caused the program to terminate.

6.5.5 Calling a procedure: Qp2SignalPase

The Qp2SignalPase() function posts an AS/400 PASE signal in the job where the API is invoked. If there is only one AS/400 PASE thread running in the job, the signal remains pending until control is transferred to the AS/400 PASE program. If other AS/400 PASE threads are running at the time Qp2SignalPase() is called, the machine may choose one of the other threads to deliver the AS/400 PASE signal.

6.5.5.1 Syntax

```
#include <qp2user.h>
   int   Qp2SignalPase(int     sign);
```

The service program is QP2USER.

6.5.5.2 Parameter

signo Input; Binary(4) signal number

signo is a non-zero value that identifies the signal to post. A positive number is interpreted as an ILE signal for which a corresponding AS/400 PASE signal should be posted. ILE and AS/400 PASE signals

correspond if they have the same name (for example, EFAULT) in a system-provided header file. A negative number is interpreted as the inverse of an AS/400 PASE signal number (providing the ability to post any arbitrary AS/400 PASE signal).

Not all ILE signals have an AS/400 PASE equivalent, and Qp2SignalPase never converts ILE SIGCHLD to a corresponding AS/400 PASE signal. This special handling for SIGCHLD avoids duplicate AS/400 PASE signals for the termination of a single child process. The reason is because the system sends both ILE and AS/400 PASE signals to the parent of any fork child process that ends.

6.5.5.3 Restrictions

Qp2SignalPase is only supported when an AS/400 PASE program is currently running in the job. This means that there must be an active invocation of the Qp2RunPase API in the job, or the job must be a forked child process.

6.5.5.4 Programming notes

Qp2SignalPase can be used directly as an ILE signal handler. However, you need to cast the function address to ignore the function result that is not used by the system for signal handlers:

```
#include <signal.h>
#include <qp2user.h>
#include <string.h>
typeahead void (SIGHANDLER*)(int);

struct sigaction        action;
memset(&action, 0, sizeof(action));
action.sa_handler = (SIGHANDLER)Qp2SignalPase;
sigaction(i, &action, 0);
```

6.5.5.5 Return value

The function result from Qp2SignalPase indicates whether the AS/400 PASE signal was successfully posted. Header file qp2user.h defines the following constants for the return code from this API:

QP2CALLPASE_NORMAL
: An AS/400 PASE signal was posted successfully.

QP2CALLPASE_ENVIRON_ERROR
: The requested function is not allowed at this time because no AS/400 PASE program is currently loaded. You can only run Qp2SignalPase while AS/400 PASE is active in the job.

QP2CALLPASE_ARG_ERROR

One or more argument values passed to the Qp2SignalPase API are invalid.

QP2CALLPASE_TERMINATING

The AS/400 PASE program is terminating. The program may have run the exit function, or a signal may have caused the program to terminate.

6.6 Debugging an AS/400 PASE application

Depending on the specifics and functions of the application, debugging an AS/400 PASE application takes several different strategies.

If the AS/400 PASE application does not require any OS/400 integration, database, or ILE functions, the application should first be debugged on AIX. Then, when moved to AS/400 PASE, a combination of AS/400 PASE dbx and AS/400 debug capabilities, VLOG and job logs, can be used. See Chapter 12, “Problem determination and messages” on page 173.

The code that is changed to use AS/400 DB2CLI, database, or ILE functions cannot be tested on AIX, but proper structure and design of the remaining parts of the application should make those parts debuggable on AIX. See Chapter 12, “Problem determination and messages” on page 173, for information on debugging an AS/400 PASE and ILE application.

6.6.1 Using dbx in AS/400 PASE

The AS/400 PASE environment supports the AIX dbx debugger utility. The utility supports debugging of related processes, such as parent and child, at the source code level if compiled as such. However, AS/400 PASE has the limitation of only being able to debug the parent or child, not both. In AIX, such an arrangement would launch another xterm window with dbx attached to the second process. AS/400 PASE does not support pty for the terminal support and, therefore, can only debug one process. Also, if using dbx in something like a shell in emacs, AS/400 PASE is unable to fool the program about being a tty, since the program is hooked up to a pipe and not stdio descriptors. The -i option is then necessary when dbx is started for interactive mode. Otherwise, the first time the Enter key is pressed without typing anything, dbx exits. This behavior is also due to no pty support. See 2.4.1, “Devices” on page 23, for more information. You could debug the parent or the child from a separate dbx session by connecting to it using the process identifier (pid).

For details on dbx, access the manual page (type in the `man` command) for the AIX machine or visit the RS/6000 & AIX Support Web site at:
`http://duke.toraix.can.ibm.com/lngfiles/dbx.html`

6.6.2 AS/400 PASE unsupported system calls

The AS/400 PASE kernel exports some system calls that are implemented by the AIX kernel but that are not supported in PASE. The default behavior for any unsupported syscall in AS/400 PASE is to write a VLOG entry and send the signal SIGILL. See Chapter 12, “Problem determination and messages” on page 173, for more information on debugging on the AS/400 system.

The unsupported syscall returns a function result of -1 with errno ENOSYS if the signal is ignored or the handler returns.

VLOG entry major/minor code 4700/000F contains this information for any unsupported system call:

- syscall number (GPR2 value)
- AS/400 PASE instruction address
- Link register value
- GPR3-10 values (if available, or zero otherwise)
- syscall name (if known, converted to uppercase)

If the VLOG was created because of an illegal instruction that was caused by the attempted use of an unsupported syscall, it is possible to scroll right to see the character representation of the VLOG information with the syscall name (the name you would put into the environment variable to work around the unsupported syscall) by pressing PF10. To scroll back to the left, use PF9.

To see the VLOG entry, programmers may need to request assistance from operations or maintenance personnel with sufficient authority to use the service tools. See 12.1.2.2, “VLOGs” on page 175, for more information on VLOGs.

AS/400 PASE applications can suppress SIGILL for unsupported system calls by setting environment variable PASE_SYSCALL_NOSIGILL in the initial AS/400 PASE environment or any time before running exec. PASE_SYSCALL_NOSIGILL is ignored if the AS/400 PASE program has the S_ISUID or S_ISGID attribute. Otherwise, it is interpreted as a list of syscall function names with optional errno values, delimited by colons. The colon-delimited values must take one of these forms:

```
syscall_name
syscall_name=errno_name     (errno_name is EINVAL, EPERM, etc.)
syscall_name=errno_number   (errno_number is 0-127)
```

SIGILL is suppressed for any syscall_name in the list that is recognized as an AS/400 PASE syscall. The first or only entry in the list may use a special syscall_name of “ALL” to set a default behavior for all unsupported syscalls. Any entry in the list that is not an AS/400 PASE syscall name is ignored, and specifying the name of a syscall that is supported by the AS/400 PASE kernel has no effect on the operation of that syscall.

Any AS/400 PASE syscall in the PASE_SYSCALL_NOSIGILL list that is unsupported by the AS/400 PASE kernel returns a function result of -1 with the specified errno value (defaulting to ENOSYS), except that specifying errno_number of 0 causes the unsupported syscall to return a function result of zero (without setting errno). An invalid errno_name or errno_number defaults to ENOSYS.

For example, the following PASE_SYSCALL_NOSIGILL value suppresses SIGILL for all unsupported syscalls. “vmount” returns EPERM and “audit” returns a function result of zero, while all other unsupported syscalls return ENOSYS:

```
export PASE_SYSCALL_NOSIGILL
PASE_SYSCALL_NOSIGILL=ALL:vmount=EPERM:audit=0
```

Note

PASE_SYSCALL_NOSIGILL is not intended to be used in shipped AS/400 products. It is provided as a convenience for product feasibility testing using unchanged AIX binaries that need to be modified for AS/400 “productization”.

Chapter 7. Database porting with AS/400 PASE

AS/400 PASE supports the OS/400 DB2 UDB Call Level Interface (CLI). Because DB2CLI on AIX and OS/400 are not proper subsets of each other, there are minor differences in a few interfaces and some APIs in one implementation that may not exist in another. This implies that the code can be generated, but *not* tested, on AIX itself. It must be tested cross-platform within AS/400 PASE. For further information on OS/400 DB2CLI, see *DB2 UDB for AS/400 SQL Call Level Interface (ODBC)*, SC41-5806. You should also refer to E.3, “Other resources” on page 222.

7.1 Data encoding considerations

The AS/400 system is an EBCDIC encoded system, by default, while AIX is based on ASCII. The difference in encoding schemes often requires data conversions between the AS/400 database and the AS/400 PASE application.

Files in the QSYS.LIB file system are always EBCDIC. That is, they can only be tagged with EBCDIC code pages. You can see the code page of a file by using the WRKLNK command, for example:

```
WRKLNK '/QSYS.LIB/ASERENA.LIB/TESTFILE.FILE/TEST.MBR')
```

Then, use option 8 to display details about the file. Files stored in other locations in the file system may be tagged with EBCDIC, ASCII, or another character set.

In the AS/400 PASE implementation of the DB2 CLI, the AS/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 AS/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.

The default AS/400 PASE CCSID is either explicitly defined or is derived based on a best match to the job CCSID. To explicitly set the AS/400 PASE CCSID, the following two environment variables must be defined and set to the appropriate values:

- `PASE_LANG`
- `QIBM_PASE_CCSID`

Extensions to libc.a give the AS/400 PASE application the ability to change the running CCSID of the application. The `_SETCCSID()` interface takes an integer of the new CCSID to switch to and returns an integer of the former CCSID. A returned value of -1 indicates an error.

Another extension gives the AS/400 PASE application the ability to override the DB2CLI interface internal conversion without changing the CCSID of the application. This function is `SQLOverrideCCSID400()` and accepts as a single parameter an integer of the override CCSID. The function returns no value, that is, it is void.

Note

The CCSID override functions, `_SETCCSID()` and `SQLOverrideCCSID400()`, must be called before any other SQLx() API for it to take effect. Subsequent invocations of these APIs are ignored.

The prototype for `_SETCCSID` is defined in as400_protos.h and exported out of libc.a in the export file as400_libc.exp. The prototype for and `SQLOverrideCCSID400` is also defined in as400_protos.h and exported out of libdb400.a in the export file libdb400.exp. See Appendix A, “Programming resources” on page 203, for information on where to get these files.

An AS/400-specific interface called SQLOverrideCCSID400() can be called before any other SQLxxx() routine to override the ASCII CCSID used when the AS/400 PASE environment is setup (Qp2RunPase(), see 6.5.3, “Invocation from an ILE application: Qp2RunPase” on page 71). The ASCII CCSID on Qp2RunPase() can only be single byte ASCII or UTF-8.

The DB2CLI library routines are in libdb400.a for the AS/400 PASE environment, and are implemented using pthread interfaces providing thread safety. The routines interface to the OS/400 service program QSQCLI to perform the desired function. A C header, sqlcli.h, and a linker export file, libdb400.exp, are used for compilation and linking on an AIX workstation. The sqlcli.h header file is available on the AS/400 system in library QSYSINC, file H, member SQLCLI. This and all other header files in QSYSINC/H can be accessed via /QIBM/include/<header>.h. See Appendix A, “Programming resources” on page 203, for information regarding where to get the other files. These files need to be placed in appropriate INCLUDE and LIBPATH directories or specified with compiler or linker option switches.

7.2 Known problems

The following known issues existed at the time this redbook was written and had not been resolved:

- Numeric string conversions done on behalf of the application by OS/400 CLI support will always result in an EBCDIC string. This is a problem if the data is being fetched from DB2 UDB into the application's variables. For example, a field in the DB is typed as an integer, but the application wants CLI to convert implicitly to a string during a fetch operation. A workaround would be for the application to do this conversion. Note that string-to-string conversions are converted correctly. Basically, a CLI application can type the data it wants in the application. If the type of the data in the database is numeric but the application types it as a string, the OS/400 database will do the conversion. Unfortunately, the conversion is to EBCDIC. If the data in the database is character and the application types it as “char”, the conversion is done correctly to ASCII.
- SQLGetSubstring always returns an EBCDIC string when sub-stringing the CLOB/DBCLOB field. The SQLGetSubString is only used for LOB data types.
- SQLTables, column 4 of the result set (table type), is always returned as EBCDIC.
- DB2 UDB doesn't support UTF-8 as of yet, exposing an issue with either mixed SBCS/DBCS text data or graphic data in the database that an application needs.

  Graphic-typed data can be rendered back to the application if the application typecasts the data as wchar. This causes the database to convert from a graphic and pure double-byte character to Unicode/UCS-2. If this type conversion isn't done, the database converts between the CCSID of the data and the CCSID of the OS/400 job. Graphic-typed data contains no shift-in and shift-out special sequences, since all of the data is expected as double bytes. Unfortunately, at this time, the database doesn't support converting between EBCDIC graphic and the CCSID (either from the Qp2RunPase() API or the SQLOverrideCCSID400() API). The solution to this problem is being investigated for inclusion in a future release.

7.3 DB2CLI example program

The following example shows how to compile the sample program on AIX and then to prepare and execute the example under AS/400 PASE.

The program uses the existing system database shipped with Client Access/400, QIWS/QCUSTCDT. The contents of the database are shown in Figure 8.

CUSNUM	LSTNAM	INIT	STREET	CITY	STATE	ZIPCOD	CDTLMT	CHGCOD	BALDUE	CDTDUE
938472	Henning	G K	4859 Elm Ave	Dallas	TX	75217	5000	3	37.00	0
839283	Jones	B D	21B NW 135 St	Clay	NY	13041	400	1	100.00	0
392859	Vine	S S	PO Box 79	Broton	VT	5046	700	1	439.00	0
938485	Johnson	J A	3 Alpine Way	Helen	GA	30545	9999	2	3987.50	33.50
397267	Tyron	W E	13 Myrtle Dr	Hector	NY	14841	1000	1	0	0
389572	Stevens	K L	208 Snow Pass	Denver	CO	80226	400	1	58.75	1.50
846283	Alison	J S	787 Lake Dr	Isle	MN	56342	5000	3	10.00	0
475938	Doe	J W	59 Archer Rd	Sutter	CA	95685	700	2	250.00	100.00
693829	Thomas	A N	3 Dove Circle	Casper	WY	82609	9999	2	0	0
593029	Williams	E D	485 SE 2 Ave	Dallas	TX	75218	200	1	25.00	0
192837	Lee	F L	5963 Oak St	Hector	NY	14841	700	2	489.50	0.50
583990	Abraham	M T	392 Mill St	Isle	MN	56342	9999	3	500.00	0

Figure 8. Data within the QIWS/QCUSTCDT database file

Complete the following steps to produce this program:

1. The header file sqlcli.h and the export file libdb400.exp are obtained and placed in the example directory ~spartz/pase/include on the AIX machine. See Appendix A, “Programming resources” on page 203, for a list of where to get these files.
2. The source file is created in ~spartz/pase on the AIX machine. The following compilation command generates the binary:

   ```
   xlc -I./include -bI:./include/libdb400.exp -o paseclidb4 paseclidb4.c
   ```

3. The resulting binary is sent via FTP to the AS/400 PASE system. In this example, it is “put” to /home/spartz/paseclidb4. See 6.3, “Accessing files from AS/400 PASE” on page 60, for methods of moving files to the AS/400 system for use by AS/400 PASE.
4. Sign on to the AS/400 PASE system. Call the QP2TERM program:

   ```
   CALL QP2TERM
   ```

5. Change the directories to where the binary was placed in IFS on the AS/400 system:

   ```
   cd /home/spartz
   ```

6. Invoke the program:

   ```
   paseclidb4
   ```

The output of this program is shown in Figure 9.

```
                          /QOpenSys/usr/bin/sh

   fun_Process(): SQLBindParam() succeeded
   fun_Process(): SQLExecute() succeeded
   fun_Process(): SQLBindCol() succeeded
   fun_Process(): SQLFetchScroll() succeeded, LastName(Alison  )
   fun_Process(): SQLFetchScroll() succeeded, LastName(Abraham )
   fun_Process(): SQLFetchScroll() completed all rows
   fun_Process(): SQLCloseCursor() succeeded
   main(): Query complete
   fun_DisConnect(): SQLDisconnect() succeeded
   fun_ReleaseDbcHandle(): SQLFreeConnect() succeeded
   fun_ReleaseEnvHandle(): SQLFreeEnv() succeeded
   main(): fun_DisConnect() succeeded
   main(): normal exit
   $

 ===>



 F3=Exit   F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
 F13=Clear F17=Top  F18=Bottom  F21=CL command entry
```

Figure 9. Output produced by the DB2CLI program sample

AS/400 PASE DB2 UDB example program source

The following source code produced the results shown in Figure 9, with the exception of the name of the AS/400 system, the user ID, and password which were changed in the fun_Connect() procedure. The appropriate values must be substituted before compilation.

```
/* File paseclidb4.c created by Daryl Spartz on Thu May 11 2000.
* AS/400 PASE DB2 UDB for AS/400 example program
 *
 * To show an example of an AS/400 PASE program that accesses
 * AS/400 DB2 UDB via SQL CLI
 *
 * Program accesses Client 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, invoke 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 */
```

Chapter 8. ILE integration with the AS/400 PASE environment

AS/400 PASE provides another program model for applications. Figure 10 shows all of the applications models now available on the AS/400 system. Traditional applications can use either old program model (OPM) or integrated language environment (ILE). Both of these models run an instruction set that is mapped into TIMI (the Machine Interface layer that shields the application from the hardware). Java applications run under the context of a Java Virtual Machine (JVM) that behaves as if it were running Java byte codes in the applications. On the AS/400 system, however, Java objects are compiled into executable objects that also use TIMI services. AS/400 PASE is different. It runs AIX binaries, and those objects use the Syscall interface to access operating system services.

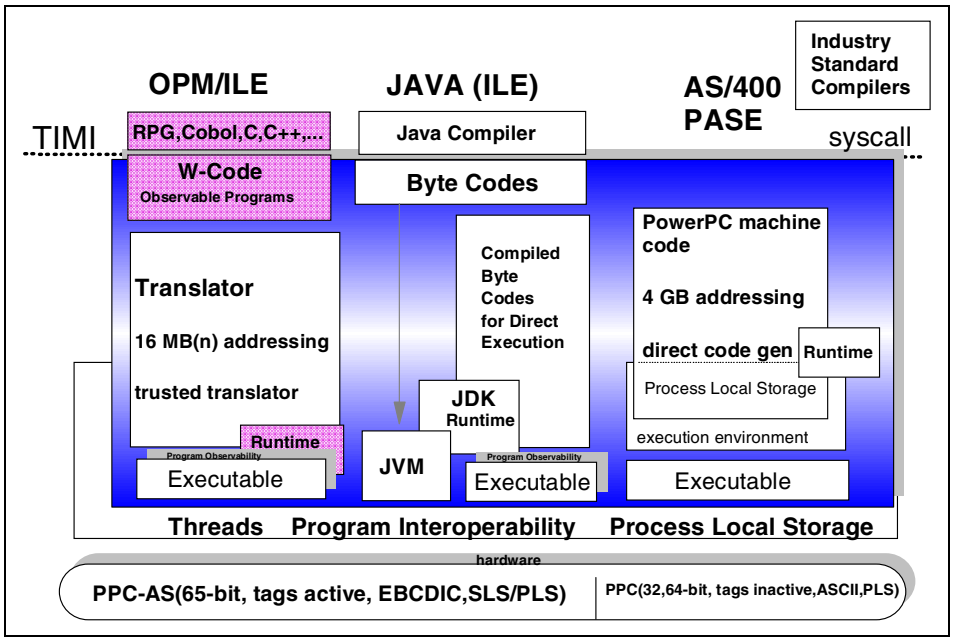


Figure 10. AS/400 program models

Some of you may wonder why we would ever want to make calls between the environments. There are several reasons you may want to access both environments, including:

- Taking advantage of legacy ILE code
- Accessing ILE middleware interfaces (such as MQSeries)
- Taking advantage of AS/400 PASE's computational speed

We assume your goal is to take full advantage of your AS/400 system. AS/400 PASE gives you the opportunity to take full advantage of both environments.

8.1 Shared addressing between AS/400 PASE and single level store

In the single level store (SLS) environment, all processes share a single address space that provides a mapping for all memory in the system (except for unnamed teraspace regions). Programs running in SLS can address all memory in the system to which the user of the program is authorized. Security and integrity are provided through a combination of page-level hardware storage protection and controlling hardware instruction sequences (so that only “safe” memory addresses are generated). Hardware instruction sequences are controlled by requiring all programs to be generated by the System Licensed Internal Code (SLIC) translator from a high-level program description called an *MI program template*.

AS/400 PASE provides a separate private address space for each process and limits the references in each user program to just memory in that address space. Programs running in AS/400 PASE can only address memory that is mapped into the address space where the program runs. They do not have direct access to the special PowerAS instructions that build tagged MI pointers. This way, an AS/400 PASE program can run any arbitrary sequence of (user-state PowerPC) hardware instructions without jeopardizing system security or integrity.

The AS/400 PASE address space provides a mapping from AS/400 PASE addresses to addresses in a teraspace region. Any memory mapped into the AS/400 PASE region of teraspace has exactly the same accessibility (relative address and storage protection) to both AS/400 PASE programs and SLS programs. Both SLS segments and unnamed memory can be mapped into teraspace. When building ILE code to be called from an AS/400 PASE application, the teraspace compile options are used.

AS/400 PASE and the SLS environment can share memory (but in a controlled manner, limited by what memory is mapped into the private address space), allowing programs to call back and forth between environments (within the context of a single process) as well as pass and share data.

Figure 11 shows the relationship between AS/400 PASE and the SLS environment. The memory for selected SLS segments is shared by mapping it into the private address space. There is a separate call stack for tag-inactive

invocations. AS/400 PASE programs deal with 4-byte (untagged) pointers, in contrast to the 16-byte tagged MI pointers used in the SLS environment.

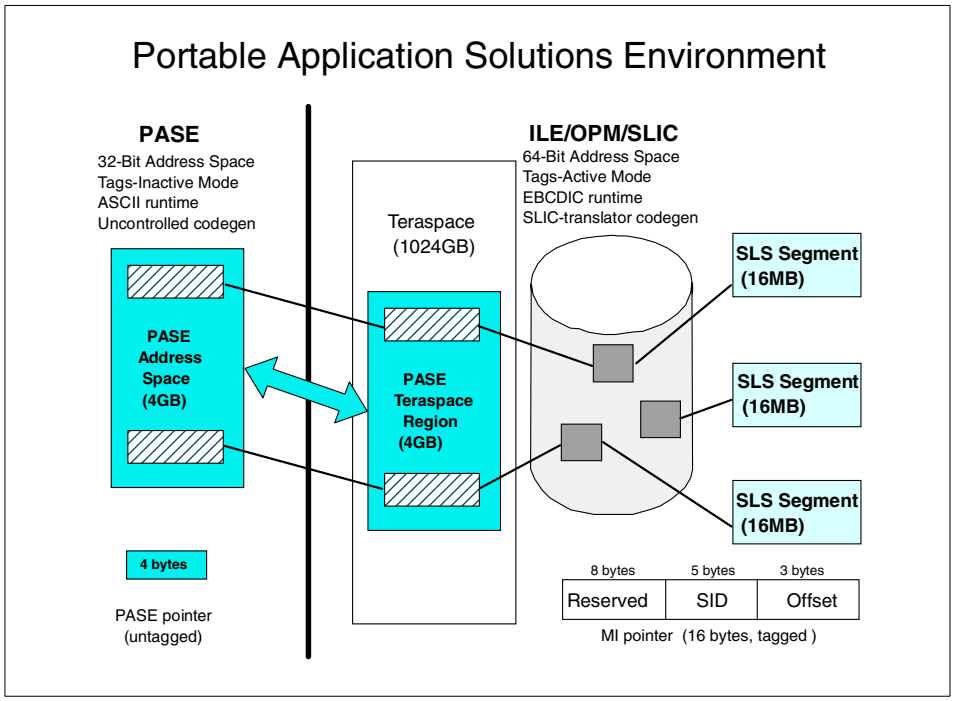


Figure 11. AS/400 PASE and the SLS environment

8.1.1 System structure

AS/400 PASE can be activated in any OS/400 job. AS/400 PASE programs are XCOFF load modules (containing PowerPC instructions and data) that are stored in bytestream files in the hierarchical directory structure of the integrated file system (IFS). AS/400 PASE is activated by launching an AS/400 PASE program using QP2SHELL or QP2TERM commands (refer to 6.5, “Starting an AS/400 PASE application” on page 65) or by a request to load or run an executable from a stream file through the Qp2RunPase API. AS/400 PASE programs can be loaded from any physical file system that supports normal stream file access (including NFS).

AS/400 PASE runtime includes a set of shared AIX libraries that an AS/400 PASE application can link to and call, as well as some ILE service programs (implementing an API to invoke AS/400 PASE and internal support functions for running an AS/400 PASE program). The shared libraries are shipped in

OS/400 option 33, while the ILE service programs are shipped in the *BASE option of OS/400.

8.1.2 Memory model and program model

The AS/400 system treats all code running in AS/400 PASE from the point of entry (the call to QP2RunPase) until the AS/400 PASE program does a system call as a single invocation. Calls back into AS/400 PASE invocations are supported and nested to any depth (subject to stack limits in SLIC, MI, and AS/400 PASE). An AS/400 PASE program can call an ILE procedure that re-invokes AS/400 PASE (using the Qp2CallPase() API) before returning. Code running in AS/400 PASE can manage stacks and heap any way it wants (within the constraints of the PowerPC Application Binary Interface used by AIX, which is the common definition for all binaries produced by AIX development tools). This is independent of how the SLS program model runs.

Any memory allocated in the private address space (including stack, heap, and any shared memory mapped into the AS/400 PASE address space) can be accessed from the SLS environment, which allows an AS/400 PASE program to pass arguments by address to ILE procedures. However, the reverse is not true. You can only pass arguments by value from ILE programs to AS/400 PASE programs.

AS/400 PASE provides support for InterProcess Communication (IPC) interfaces (shared memory, messages, and semaphores accessed through the shm, msg, and sem APIs) using the same system support as corresponding ILE interfaces.

Note

AS/400 PASE, like ILE, currently provides support for mapping keyed memory regions using the shm interfaces, rather than files. IBM intends to provide file services and the mmap family of APIs in a future release.

When an AS/400 PASE program calls an ILE procedure, pointer arguments are converted by SLIC to an equivalent teraspace address in an MI pointer so the ILE procedure can reference the same memory as that addressed by the AS/400 PASE pointer.

The AS/400 PASE memory model is uniform, with no restrictions on accessing structures that span any range of bytes in the address space. The traditional AS/400 SLS memory model is segmented with hardware and program model (translator-generated) checks to prevent memory accesses

that span 16 MB segment boundaries. MI programs built with the TERASPACE(*YES) attribute can access all AS/400 PASE memory, with the same storage protection as the AS/400 PASE program.

The AS/400 PASE program model does not support MI exceptions or MI events. All synchronous and asynchronous interrupts reported to an AS/400 PASE program are delivered as signals (signals are a simpler and potentially faster model than MI exceptions).

AS/400 PASE programs generally must expect and tolerate interruption or termination at any point during execution, since there are no pre-defined interruption points in the hardware architecture they use. An AS/400 PASE program can register signal handlers to trap interrupts, including the ability to register a SIGTERM signal handler for cleanup processing prior to “unexpected” AS/400 PASE termination (for example, when MI exception handling causes AS/400 PASE to be terminated).

Any condition that would normally cause an MI exception to be delivered to the invocation (if it were an MI program invocation) is delivered to an AS/400 PASE program as a signal. This includes program-error interrupts within the AS/400 PASE program (such as divide by zero), as well as exception percolation from newer (MI or SLIC) invocations to an older AS/400 PASE invocation.

The system delivers MI event signals and (asynchronous) POSIX/ILE signals to a process or thread while that thread is running an AS/400 PASE program. It also allows MI process management functions (such as terminate process) to interrupt AS/400 PASE. This means that an AS/400 PASE program may be interrupted at any point to invoke an MI program or ILE procedure to handle an MI event or POSIX/ILE signal. The handler can resume the interrupted AS/400 PASE program by simply returning. Control may not return to the interrupted AS/400 PASE program (although a SIGTERM handler may be run) if the event handler does something that cancels the AS/400 PASE invocation (including the possibility of terminating the process).

AS/400 PASE signals and POSIX/ILE signals are independent, so it is not possible to directly invoke a handler for one signal type by raising the other type of signal. However, the Qp2SignalPase() API can be used as the handler for any ILE signal to post a corresponding AS/400 PASE signal to the AS/400 PASE program. The AS/400 PASE program can also define handlers for AS/400 PASE signals that call ILE procedures to post equivalent ILE signals. Program QP2SHELL and the AS/400 PASE fork function always setup handlers to map every ILE signal to a corresponding AS/400 PASE signal.

Debug support for AS/400 PASE programs is provided by the dbx utility (ported from AIX). dbx provides both source-level and machine-level debugging capabilities. Other debuggers for AS/400 PASE programs may be supported in the future. IBM service personnel also have access to internal tools, such as SLIC native macros, which may help to debug problems that could be in user code or system code. Refer to 6.6.1, “Using dbx in AS/400 PASE” on page 84, for further information.

8.1.3 File system and socket support

AS/400 PASE runtime normally uses ILE C runtime support for files `stdin`, `stdout`, and `stderr`, which provide consistent behavior for AS/400 PASE and ILE programs. However, AS/400 PASE must always support `stdin`, `stdout`, and `stderr` as file descriptors 0, 1, and 2, while ILE C does not always use file descriptors for `stdin`, `stdout`, and `stderr`. The AS/400 PASE loader needs to open bytestream files for AS/400 PASE executables, which must not consume descriptor numbers that are visible to the AS/400 PASE program. System support for AS/400 PASE meets these constraints with mapping between AS/400 PASE file descriptors and IFS file descriptors. The result is that different descriptor numbers are used by AS/400 PASE programs and ILE C programs to access the same open file. An AS/400-unique extension to the fstatx function, STX_XPFFD_PASE, allows an AS/400 PASE program to determine the IFS descriptor number for an AS/400 PASE file descriptor. Special values (negative numbers) are returned for any AS/400 PASE descriptor attached to ILE C runtime support for files `stdin`, `stdout`, and `stderr`.

If the ILE environment variable QIBM_USE_DESCRIPTOR_STDIO is set to Y or I when the Qp2RunPase() API is invoked, AS/400 PASE synchronizes file descriptors 0, 1, and 2 with IFS so both AS/400 PASE and ILE C programs use the same descriptor numbers for files `stdin`, `stdout`, and `stderr`. When operating in this mode, if either AS/400 PASE code or ILE C code closes and/or reopens file descriptor 0, 1, or 2, the change affects `stdin`, `stdout`, and `stderr` processing for both environments.

AS/400 PASE runtime generally does no character encoding conversion for data read or written through AS/400 PASE file descriptors (including sockets), except that ASCII/EBCDIC conversion is done (between the AS/400 PASE CCSID and job default CCSID) for data read from ILE C `stdin` or written to ILE C `stdout` and `stderr`.

Note

There are two environment variables that control the automatic translation of `stdin`, `stdout`, and `stderr`:

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

ASCII/EBCDIC conversion for AS/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.

Support to synchronize AS/400 PASE and IFS file descriptors (`QIBM_USE_DESCRIPTOR_STDIO`) and support for binary `stdin`, `stdout`, and `stderr` processing is included.

8.1.4 Runtime support

The breadth of AS/400 PASE runtime support shipped with OS/400 (in option 33) has been greatly extended in V4R5 with support, including X-windows with Motif, several shells, and a large number of utilities. Applications running in AS/400 PASE work in ASCII. The AIX C compiler, xlc, does not support EBCDIC, and much of the AS/400 PASE runtime comes from AIX code that only works in ASCII. Any AS/400 PASE runtime service that the system provides (including any system call or runtime function in a shared library shipped with OS/400 option 33) handles ASCII/EBCDIC conversions as needed, although generally no conversions are done for data read or written to a file descriptor (bytestream file or socket). AS/400 PASE programs pass ASCII (or UTF-8) path names to the open function to open byte stream files, where the name is automatically converted to EBCDIC, but any data read or written from the open file is *not* converted. See 6.4, “Configuration tips” on page 62, for a tip on case sensitivity.

It is up to applications running in AS/400 PASE to handle character encoding conversions for strings for calls from AS/400 PASE to arbitrary ILE procedures. AS/400 PASE runtime support includes the iconv_open, iconv, and iconv_close functions for character encoding conversion.

Note

AS/400 PASE and ILE have independent implementations of *iconv* interfaces, each with its own translation tables. The translations supported by AS/400 PASE iconv support can be modified and extended by users because they are stored as bytestream files stored in IFS.

Conversion objects and locales for AS/400 PASE are stored as byte stream files in IFS. A number of conversion objects and locales ship in OS/400 option 33, and customers may augment this set with their own files. All AS/400 PASE locales use ASCII or UTF-8 character encoding, so all AS/400 PASE runtime works in ASCII (or UTF-8).

Note

Locale support for AS/400 PASE is independent of either forms of locale support used by AS/400 ILE C programs (object types *CLD and *LOCALE). In addition to internal structure differences, none of the existing shipped locales for ILE C programs support ASCII.

AS/400 PASE runtime only provides stream file access to or through the IFS, which is the equivalent of SYSIFCOPT(*IFSIO) for ILE C code. Access to the DB2/400 database is provided through the SQL CLI functions exported by an AS/400 PASE shared library included with OS/400 option 33. An AS/400 PASE program that requires access to object types or interface options that are not supported through IFS or SQL CLI can directly call ILE procedures.

AS/400 PASE runtime is based on AIX runtime and handles syntactic differences (such as ASCII/EBCDIC) between AS/400 PASE and the ILE environment. However, there are some functional limits and behavior differences between AS/400 PASE and AIX runtime (generally beneath the level of POSIX specification) for services, such as directory, file system, and sockets, which AS/400 PASE runtime implements as mappings to services that run in the AS/400 ILE environment.

Environment variable support for AS/400 PASE exists independently of the ILE runtime. The system does not implicitly set any AS/400 PASE environment variables. However, the Qp2RunPase() API allows its caller to specify a set of environment variables to initialize in AS/400 PASE, and program QP2SHELL passes a copy of all ILE environment variables to the AS/400 PASE program. Environment variables prefixed with the string

“PASE_” are presented to the AS/400 PASE environment in the original format and with the prefix removed. Conflicts with identical ILE variables will be resolved by using the “PASE_”’ prefix exclusively. As QP2TERM uses QP2SHELL, it also exhibits this behavior.

SLIC implements support for the (AIX) system calls needed to run the C library runtime subset supported by AS/400 PASE, and also supports system calls for platform-specific functions, such as building a tagged space pointer (_SETSPP) and calling an ILE procedure (_ILECALL). Requirements for new platform-specific system calls should be identified to IBM through PartnerWorld for Developers, your local AS/400 Support Center, or the Custom Technology Center (CTC).

8.1.5 Development environment

AS/400 PASE development requires an AIX 4.3.3 system for source code generation to run the compiler, xlc or xlC, and linking directive ID, ld command. The AIX assembler for PowerPC can also be used.

8.1.6 Performance

AS/400 PASE programs do not incur overhead to manage tagged pointers, MI boundaries, and the MI exception architecture, so low-level code generation may be more efficient than ILE in some cases. For example, incrementing a pointer can be done with a single instruction instead of the multi-instruction sequence required for MI, and a form of setjmp/longjmp that does not affect signal handling is provided that does not do system calls.

Using a segment table for address translation increases the process working set and adds additional work in the hardware for address translation. However, the processors used in AS/400 systems contain Segment Lookaside Buffer (SLB) entries or segment registers to speed address translation. Address translation through an SLB entry or segment register is as fast as direct translation (used for tags-active mode). AS/400 PASE programs should exhibit the locality of a reference similar to AIX (contributing to the effectiveness of SLB or segment register hardware).

Generally, we advise that you store your application binaries in the local stream file system. It is much slower to invoke PASE programs if your binaries (base program and libraries) are outside of the local stream file system since file mapping cannot be done.

8.2 Calling Java from AS/400 PASE

To give you another example of the integration of PASE with AS/400 facilities, the following code shows Java being called from AS/400 PASE:

```
#include <stdio.h>
#include <stdlib.h>
#include "as400_protos.h"

/*****************************************
 Sample: AS/400 PASE program which
  invokes a Java class method.
  Compile in the following way:
  xlc -bI:/home/myExpDir/as400_libc.exp
       -I/home/myIncludeDir/ JavaHello.C -o Javahello
  Replace the directory names with the location
  of the exports and the as400_protos.h and
  as400_types.h files. These files must be
  obtained from the PartnerWorld Web site,
  or AS/400 Software Support.
*****************************************/

int main(int argc, char *argv[])
{
  /* print out a greeting to confirm entry into AS/400 PASE program */
  printf("Hello I am in an AS/400 PASE program\n");

  int result;
  char cmd[39];

   sprintf(cmd, "JAVA CLASS('com.ibm.as400.system.Hello')");
   result = systemCL(cmd, SYSTEMCL_MSG_STDOUT);

  return 0;

}
```

8.3 Doing callouts to ILE from AS/400 PASE

In this section, we discuss the primary method for calling ILE code from AS/400 PASE and provide a simple example for you to follow when learning to call from AS/400 PASE to ILE applications.

Before we begin our examples, we need to set a few rules. All ILE programs that will be called from AS/400 PASE must be compiled with the teraspace option set to *YES. If your ILE programs are not compiled this way, you will get a vague error in your AS/400 PASE application, for example: SIGTERM with no error. If your ILE code being called is part of a service program (which our sample is), all the modules in the service program (and any programs) must be compiled with the teraspace option set to *YES.

Text being passed between ILE and AS/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. See 4.5, “Character sets and terminal I/O” on page 43, for additional information on CCSIDs and AS/400 PASE.

There are several methods for calling ILE code from within AS/400 PASE and each method has its own advantages and disadvantages. Before starting, you need to determine whether you want to create and initialize your own pointers to the ILE code or use our methods. We recommend using our pre-canned methods but provide a generic sample for you to create your own.

In the example, we show two separate ways of calling the ILE procedure from an AS/400 PASE application. Each of these sections build on each other. Therefore, we recommend that you use the final section for testing your skills at calling ILE from AS/400 PASE.

Note

When calling from the AS/400 PASE environment to the OS/400 environment, you must make sure that the OS/400 program is compiled with *CALLER for activation group. Many AS/400 CL commands use programs that were created with activation group *NEW and, when they end or return, they destroy their activation group. When that activation group ends in a multi-thread capable job (such as any OS/400 job started by the AS/400 PASE `fork()` function), the AS/400 PASE program that called it will also end.

In short, use AS/400 CL commands and ILE programs that are created or compiled with the activation group *CALLER.

8.3.1 Setting up variables and structures

Some set up of variables and structures is required for calling out of AS/400 PASE to ILE. In particular, you must set up a “signature”, “result type”, and an “argument list” variable.

8.3.1.1 Header files

The header files as400_types.h, and as400_protos.h used in the following samples are not on your AS/400 system. You need to obtain these header files from the PartnerWorld Web site (see Table 12 on page 203). If the files are not there, obtain them from your AS/400 Software Support organization. The header files are:

```
#include <as400_types.h>
#include <as400_protos.h>
```

The last two headers (as400_types.h and as400_protos.h) are critical to your AS/400 PASE program. The as400_type.h header file contains the definition of the types used throughout the definition of the structures and variables in your AS/400 PASE program.

8.3.1.2 Signature

The signature structure contains a description of the sequence and types of arguments passed between AS/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 `#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 4-byte float arguments to be widened to 8 bytes.

```
static arg_type_t signature[]= {
   ARG_INT32...
   . . .
   . . .
};
```

8.3.1.3 Result type

The result type is straightforward and works much like a return type in C:

```
static result_type_t result_type = RESULT_INT32;
```

8.3.1.4 Argument list

If you build your own argument list, it list should be structurally the same as the signature, except it will actually have variables assigned to its structure. However, you can use size_ILEarglist() and build_ILEarglist() to dynamically build the argument list based on the signature. The following arglist is used in the second sample, but not the first:

```
typedef struct {
   . . .
   . . .
} ILEarglist;
```

8.3.1.5 ROUND_QUAD function

To guarantee that your pointers are quadword aligned, you can use the ROUND_QUAD macro. However, you must use `#define ROUND_QUAD` before using it in your AS/400 PASE program:

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

8.3.1.6 AS/400 PASE functions for accessing ILE

AS/400 PASE provides a number of functions for accessing ILE code. Which ones you use depends upon how much preparation and structure building you want to do yourself versus how much you want the compiler to do for you. The common AS/400 PASE functions are:

- **size_ILEarglist**: Sizes an ILE argument list
- **build_ILEarglist**: Builds an ILE argument list
- **_ILELOAD**: Loads an ILE bound program
- **_ILESYM**: Locates an exported ILE symbol
- **_ILECALL**: Calls an ILE procedure
- **_MEMCPY_WT** and **_MEMCPY_WT2**: Copy memory with tags
- **_SETSPP**: Sets space pointer
- **_CVTSPP**: Converts space pointer
- **_SETCCSID**: Sets PASE runtime CCSID
- **systemCL**: Runs a CL command

In our samples, we only use `_ILELOAD`, `__ILESYM`, `size_ILEarglist`, `build_ILEarglist`, and `_ILECALL`. However, each API is thoroughly described in its own section following the sample code sections.

8.3.2 A two part sample for calling ILE from AS/400 PASE

In this section, we present an actual working sample of AS/400 PASE code making a call to an ILE procedure that is part of a service program. 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 “simple” procedure demonstrates building your data structures for the _ILECALL using our pre-canned methods. The “best” procedure then builds the argument list manually.

Note

We recommend calling ILE procedures or modules that are part of a service program rather than part of a program object. We recommend this because _ILELOAD works better with *SRVPGMs than it does with *PGMs.

8.3.2.1 AS/400 PASE C code

Interspersed in the following sample code are comments that explain the code. Make sure to read these comments as you enter or review the sample. Remember, as400_types.h and as400_protos.h are not shipped with AS/400 PASE. You need to acquire them from PartnerWorld for Developers, the AS/400 Support Center near you, or the CTC.

```
/* 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 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 (re)initialization
 *  after exec()
 */
pid_t  init_pid = -1;
ILEpointer*ILEtarget;/* pointer to ILE procedure */

/*
 *  ROUND_QUAD finds a 16-byte (quadword) 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 guarrantee 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 and/or
 *  floating point arguments shorter than 8 bytes
 *  implies that the target ILE C procedure is compiled
 *  with #pragma argument(ileProcedureName, nowiden)
 *
 *  Absent this pragma, standard C linkage for ILE
 *  requires 1-byte and 2-byte integer arguments be
 *  widened to 4-bytes and 4-byte float arguments 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/declared structure
 *  for the ILE argument list. However, this wrapper
 *  is simpler to understand and code, because it performs
 *  a malloc to obtain storage. If an exception or
 *  signal occurs, the storage may not be feed. If your
 *  program needs to prevent such a wrap storage leak,
 *  a signal handler must be built to handle it, or you can
 *  use the methods in wrapper2.
 */
int wrapper_1(int arg1, void *arg2, double arg3,
                            char arg4, aggregate arg5, short arg6)
{
    int result;
```

```
    /*
     *  xlc does not guarrantee 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.  The last argument for build_ILEarglist is
     *  ignored/unused because the function result for this
     *  ILE procedure is not a structure or union
     */
    build_ILEarglist(ILEarglist,
            &arg1,
            signature,
            result_type,
            0);

    /*
     *  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/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;
}

/*
 *  ILEarglist 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 don't 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/declared structure for the
 *  ILE argument list to improve execution efficiency and
 *  avoid heap storage leaks if an exception/signal occurs
 */
int wrapper_2(int arg1, void *arg2, double arg3,
                        char arg4, aggregate arg5, short arg6)
{
    /*
     *  xlc does not guarrantee 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/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);
    }
  }
}
```

8.3.2.2 ILE C code

You now write the ILE C code for this sample on your AS/400 system. You need a source physical file (SRCPF) in your library in which to write the code. Again, in the ILE sample, you will find comments 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 strictly necessary */

/*
 *  The arguments and function result for this ILE procedure
 *  must be equivalent to the values presented to _ILECALL
 *  function in the AS/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;
}
```

8.3.2.3 AS/400 DDS

For this sample, you need a simple database file. We created an externally described file and populated it with one record.

```
R PASEDATA
CHARDTA       25A
```

8.3.2.4 Compilation and runtime commands

When you compile your AS/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/rchland.ibm.com/usr1/shupe/PASE/as400_libc.exp
         PASEtoILE.c
```

When you compile your ILE C module and service program, you must make sure to compile them with the teraspace option. Otherwise, AS/400 PASE can not 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)
```

8.3.2.5 Porting and testing your modules

To access your code from AIX, you need to FTP it to your AS/400 system. From a 5250 command line, type:

```
CALL QP2TERM
```

This puts you in the PASE command shell. Now enter:

```
mkdir /home/PASEtoILE
cd /home/PASEtoILE
```

Back on your AIX terminal session, perform these steps:

1. `cd <where you put your source>`
2. `ftp <your as400>`
3. Sign on to your AS/400 system.
4. `cd /home/PASEtoILE`
5. `bin`
6. `put PASEtoILE`
7. `quit`

Since the first ftp command's file name started with "/", the file was put into the stream file part of the file system.

The AS/400 PASE program is now in the /home/PASEtoILE directory on your AS/400 system. From your QP2TERM session, enter on the command line:

```
PASEtoILE
```

If you made no errors entering the code, you should see the output from the program as shown in Figure 12 on page 120.

```
                              /QOpenSys/usr/bin/sh

  $
> cd /home/shupe
  $
> PASEtoILE
  The results of the simple wrapper is: Sample 1 output text

  The results of best_wrapper function is: First record

  $




===>



F3=Exit   F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
F13=Clear F17=Top  F18=Bottom  F21=CL command entry
```

Figure 12. PASEtoILE output

However, if you have errors, you need to perform some debugging. Refer to Chapter 12, “Problem determination and messages” on page 173, for an in-depth discussion of debugging this sample program.

8.3.3 size_ILEarglist() function

The size_ILEarglist function computes the size of an ILE argument list from the function signature information.

8.3.3.1 Syntax

```
#include <as400_types.h>
#include <as400_protos.h>

size_t
size_ILEarglist(arg_type_t       *signature);
```

Libraries: Standard C Library (libc.a)

8.3.3.2 Parameters

signature Input; Pointer to a list of arg_type_t values
signature is the address of a list of arg_type_t values that specify the sequence and type of all argument values passed to the ILE procedure. The number of arguments processed by the

size_ILEarglist function is determined by the number of entries in the signature list, which is determined by the location of the first ARG_END value in the list. The following values are supported in the signature list:

- **ARG_END**: Specifies the end of the signature list
- **ARG_INT8**: Signed 1-byte integer argument
- **ARG_UINT8**: Unsigned 1-byte integer argument
- **ARG_INT16**: Signed 2-byte integer argument
- **ARG_UINT16**: Unsigned 2-byte integer argument
- **ARG_INT32**: Signed 4-byte integer argument
- **ARG_UINT32**: Unsigned 4-byte integer argument
- **ARG_INT64**: Signed 8-byte integer argument
- **ARG_UINT64**: Unsigned 8-byte integer argument
- **ARG_FLOAT32**: 4-byte floating-point argument
- **ARG_FLOAT64**: 8-byte floating-point argument
- **ARG_MEMPTR**: The argument is an ILE pointer value for which the AS/400 PASE program specifies an AS/400 PASE memory address.
- **ARG_SPCPTR**: The argument is an ILE pointer value for which the AS/400 PASE program specifies an MI Space Pointer.
- **ARG_OPENPTR**: The argument is an ILE pointer value for which the AS/400 PASE program specifies an MI pointer of any type.
- **Any value in the range 1 to 32767**: The argument is an aggregate (structure or union). The signature list entry value is the length (in bytes) of the aggregate.

8.3.3.3 Return value

The function result from size_ILEarglist is the number of bytes required to build the ILE argument list (including storage for the ILEarglist_base type) or zero if an error was detected in the signature list of argument type values.

8.3.4 build_ILEarglist() function

The build_ILEarglist function builds an ILE argument list from information about the function argument, result types, and argument values copied from an AS/400 PASE function with the same signature.

This function encapsulates sufficient knowledge of parameter linkage conventions for both the AIX PowerPC ABI and MI architecture to copy or convert argument values from an AS/400 PASE argument list to the form required by an ILE procedure.

8.3.4.1 Syntax

```
#include <as400_types.h>
#include <as400_protos.h>
size_t
build_ILEarglist(ILEarglist_base    *ILEarglist,
                 void               *PASEarglist,
                 arg_type_t         *signature,
                 result_type_t       result_type,
                 void               *result_buf);
```

Libraries: Standard C Library (libc.a)

8.3.4.2 Parameters

`ILEarglist` Output; Pointer to buffer for ILE argument list

ILEarglist is the address of a 16-byte aligned memory buffer allocated by the caller where the ILE argument list should be built. The buffer must be large enough to contain all the arguments specified by the signature list. The size_ILEarglist function can be used to determine how much storage is needed before calling build_ILEarglist.

The build_ILEarglist function does not always modify or set every byte of the output ILE argument list, so the caller may want to set the buffer to zero (0) the buffer before calling this function.

`PASEarglist`

Input; Pointer to an AS/400 PASE argument list

PASEarglist is the address of the first argument in the argument list passed to an AS/400 PASE function that accepts arguments specified by the signature list and returns a result of the type specified by the result_type argument.

> **Note**
>
> The AIX PowerPC ABI passes some argument values in registers, but all argument values are forced into memory if the called function takes the address of its first argument.

signature Input; Pointer to a list of arg_type_t values

signature is the address of a list of arg_type_t values that specify the sequence and type of all argument values passed to the ILE procedure. The number of arguments processed by the build_ILEarglist function is determined by the number of entries in the signature list, which is determined by the location of the first ARG_END value in the list. The following values are supported in the signature list:

- **ARG_END**: Specifies the end of the signature list
- **ARG_INT8**: Signed 1-byte integer argument
- **ARG_UINT8**: Unsigned 1-byte integer argument
- **ARG_INT16**: Signed 2-byte integer argument
- **ARG_UINT16**: Unsigned 2-byte integer argument
- **ARG_INT32**: Signed 4-byte integer argument
- **ARG_UINT32**: Unsigned 4-byte integer argument
- **ARG_INT64**: Signed 8-byte integer argument
- **ARG_UINT64**: Unsigned 8-byte integer argument
- **ARG_FLOAT32**: 4-byte floating-point argument
- **ARG_FLOAT64**: 8-byte floating-point argument
- **ARG_MEMPTR**: The argument is an ILE pointer value into which the caller has stored an AS/400 PASE memory address (in member field address). The build_ILEarglist function converts the AS/400 PASE memory address to an equivalent single-level store address, which it passes to the ILE procedure. The build_ILEarglist function generally updates the ILE pointer argument value in memory so it contains a tagged MI space pointer (but the memory may not be updated if the target ILE procedure uses ARGOPT linkage).

Note

build_ILEarglist does not support argument type values ARG_SPCPTR and ARG_OPENPTR (which are supported by _ILECALL) because standard PASE (AIX PowerPC ABI) linkage cannot ensure 16-byte alignment for arguments (necessary to preserve tags for ILE pointers).

result_type
Input; result_type_t

result_type specifies the type of function result returned by the ILE procedure. The following values are supported:

- **RESULT_VOID (zero)**: No function result, or an aggregate function result
- **RESULT_INT8**: Signed 1-byte integer result
- **RESULT_UINT8**: Unsigned 1-byte integer result
- **RESULT_INT16**: Signed 2-byte integer result
- **RESULT_UINT16**: Unsigned 2-byte integer result
- **RESULT_INT32**: Signed 4-byte integer result
- **RESULT_UINT32**: Unsigned 4-byte integer result
- **RESULT_INT64**: Signed 8-byte integer result
- **RESULT_UINT64**: Unsigned 8-byte integer result
- **RESULT_FLOAT64**: 8-byte floating-point result

MI architecture specifies that an aggregate function result is returned by an ILE procedure into a buffer that is allocated by the caller and passed to the target procedure using a special field in the argument list. If the function result is an aggregate, build_ILEarglist sets the result.r_aggregate.addr field (in type ILEarglist_base) to the address of the AS/400 PASE memory buffer for the function result, and the result_type passed to this function should be zero.

result_buf Input; Pointer to result buffer

result_buf specifies the address of a buffer that will be used for an aggregate function result no longer than four bytes. The result_buf value is ignored (can be zero) for function results other than aggregates (which are returned in the ILE argument list structure) and for aggregates longer than four bytes (which use a buffer allocated by the caller of the AS/400 PASE function and passed as a hidden argument, per the AIX PowerPC ABI).

8.3.4.3 Restrictions

The build_ILEarglist function does no character encoding conversions, so the AS/400 PASE program may need to convert argument and result character

strings between ASCII/EBCDIC. The AS/400 PASE runtime function `iconv` can be used for character conversions.

8.3.4.4 Return value

The function result from build_ILEarglist is the number of bytes required to build the ILE argument list (including storage for the ILEarglist_base type), or zero if an error was detected in the input arguments.

8.3.5 _ILELOAD() function

The _ILELOAD function loads a bound program into the ILE activation group associated with the procedure that launched AS/400 PASE (either the caller of Qp2RunPase or the activation group created for program QP2FORK in the child process of a fork).

8.3.5.1 Syntax

```
 #include <as400_types.h>
 #include <as400_protos.h>
int
 _ILELOAD(char               *path,
         unsigned int        flags);
```

Libraries: Standard C Library (libc.a)

8.3.5.2 Authorities and locks

_ILELOAD requires standard authority to the directories or OS/400 library objects needed to locate the bound program. It also requires the same authority as the QleActBndPgm API to the bound program object.

8.3.5.3 Parameters

path Input; Pointer to the name of the bound program

path is the address of a null-terminated character string that contains either an IFS path name or a qualified library or object name for the bound program, depending on the value of the flags argument. The character string must be encoded in the (ASCII) CCSID value specified on the Qp2RunPase API or the last successful invocation of the _SETCCSID function.

flags Input; unsigned int

flags specifies options to control how the bound program is located and activated. The flags value is a bitwise logical-or of the following values:

- **ILELOAD_PATH (zero)**: Specifies that the path argument contains an absolute or relative path in the IFS namespace. Alphabetic case is either ignored or honored depending on the attributes of the parts of the IFS namespace being searched. ILELOAD_PATH and ILELOAD_LIBOBJ are mutually exclusive.
- **ILELOAD_LIBOBJ**: Specifies that the path argument contains a qualified library or object name of a service program (where omitting the library name implies resolving to the object through the job library list). Alphabetic case is honored when searching for a library or object name (so the string should be all uppercase). ILELOAD_PATH and ILELOAD_LIBOBJ are mutually exclusive.

8.3.5.4 Restrictions

_ILELOAD uses the QleActBndPgm API to activate the bound program, so it includes any restrictions for that API.

8.3.5.5 Return value

If the bound program was successfully activated (including the case where it was already activated before _ILELOAD ran), the function result is an activation mark that uniquely identifies the activation within the process.

A function result of -1 indicates an error that is further qualified by an errno value. These errno values can be returned, with other values also possible (such as AS/400-unique ILE errno EDAMAGE):

- **EACCESS**: Not authorized to a library or directory that is needed to resolve the specified path
- **EBUSY**: A library or directory needed to resolve the specified path is currently in use (locked)
- **EFAULT**: A memory fault occurred attempting to de-reference the path argument string
- **EINVAL**: An invalid argument value was specified
- **EINTR**: A signal interrupted the operation
- **ENAMETOOLONG**: Some component of the specified path is too long, or the entire path string exceeds the system limit
- **ENOENT**: No file or object was found for the specified path
- **ENOTDIR**: A qualifier part of the IFS path is not a directory
- **ELOOP**: Too many levels of symbolic links

8.3.5.6 Error conditions

Memory errors and errors during ILE program activation may be reported with exceptions converted to AS/400 PASE signals (not return code or errno values).

The specific AS/400 PASE signal delivered for an MI exception depends on the OS/400 message identifier for the exception. AS/400 PASE delivers the same signal as ILE C for MCH and CPF message IDs (see the *ILE C/400 Programmer's Reference,* SC41-5607, for details), and uses SIGTERM for any exception that is not an MCH or CPF message ID. AS/400 PASE and ILE use different signal numbers, but AS/400 PASE mapping uses the same signal condition name (such as SIGSEGV) that is used by ILE C for each message ID.

An AS/400 PASE signal handler can determine whether a signal is associated with an exception message by inspecting the first word (4 bytes) immediately following the struct sigcontext (as defined on AIX) argument passed to the handler. A non-zero value is the message reference key for the OS/400 message that produced the signal. The AS/400 PASE program can pass the message reference key value to an ILE procedure that receives the exception message for more details about the error.

8.3.6 _ILESYM() function

The _ILESYM function locates an exported symbol in the activation of an ILE bound program and returns addressability to the data or procedure associated with the symbol.

8.3.6.1 Syntax

```
#include <as400_types.h>
#include <as400_protos.h>
int
_ILESYM(ILEpointer          *export,
        int                  actmark,
        char                 *symbol);
```

Libraries: Standard C Library (libc.a)

8.3.6.2 Authorities and locks

_ILESYM requires the same authority as the QleGetExp API.

8.3.6.3 Parameters

export Input; Pointer to a buffer for a tagged pointer

export is the address of an (aligned) 16-byte buffer that is used to store a tagged pointer to the data or procedure associated with the exported symbol.

actmark Input; int

actmark specifies an activation mark value that identifies the activation that is searched for the symbol. An input of zero (0) causes the system to search all activations in the activation group that launched AS/400 PASE (either the activation associated with the activation group that ran the Qp2RunPase API or the activation group created for program QP2FORK). The _ILELOAD interface returns an activation mark value when it loads a bound program.

symbol Input; Pointer to the name of the symbol

symbol is the address of a null-terminated character string that specifies the name of the symbol to be located. The character string must be encoded in the (ASCII) CCSID value specified on the Qp2RunPase API or the last successful invocation of the _SETCCSID function.

8.3.6.4 Restrictions

_ILESYM uses the QleGetExp API to get a pointer to the exported symbol, so it includes any restrictions for that API.

8.3.6.5 Return value

If the symbol was successfully located, the export pointer is set to the address of the function or data associated with the symbol, and the function result is set to one of these values:

- **ILESYM_PROCEDURE**: Specifies that the export argument return value is a pointer to an ILE procedure. An ILE procedure pointer can be used with the _ILECALL function to call the ILE procedure.
- **ILESYM_DATA**: Specifies that the export argument return value is a tagged space pointer to a data item in the ILE activation.

A function result of -1 indicates an error that is further qualified by an errno value. These errno values can be returned, with other values that are also possible (such as AS/400-unique ILE errno EAPAR):

- **EACCESS**: Not authorized to the activation identified by the actmark argument value.
- **ENOENT**: The symbol was not found in the specified activation.

8.3.6.6 Error conditions

Memory errors and errors during ILE symbol resolution process may be reported with exceptions converted to AS/400 PASE signals (not return code or errno values).

The specific AS/400 PASE signal delivered for an MI exception depends on the OS/400 message identifier for the exception. AS/400 PASE delivers the same signal as ILE C for MCH and CPF message IDs (see the *ILE C/400 Programmer's Reference,* SC41-5607, for details), and uses SIGTERM for any exception that is not an MCH or CPF message ID. AS/400 PASE and ILE use different signal numbers, but AS/400 PASE mapping uses the same signal condition name (such as SIGSEGV) that is used by ILE C for each message ID.

An AS/400 PASE signal handler can determine whether a signal is associated with an exception message by inspecting the first word (4 bytes) immediately following the struct sigcontext (as defined on AIX) argument passed to the handler. A non-zero value is the message reference key for the OS/400 message that produced the signal. The AS/400 PASE program can pass the message reference key value to an ILE procedure that receives the exception message for more details about the error.

8.3.7 _ILECALL() function

The _ILECALL function calls an ILE procedure from an AS/400 PASE program. This API transfers control to an ILE procedure specified by a (16-byte tagged) ILE procedure pointer, passing arguments and returning the function result (specified by arguments passed to _ILECALL).

8.3.7.1 Syntax

```
#include <as400_types.h>
#include <as400_protos.h>
int
 _ILECALL(ILEpointer         *target,
          ILEarglist_base    *ILEarglist,
          arg_type_t         *signature,
          result_type_t       result_type);
```

Libraries: Standard C Library (libc.a)

8.3.7.2 Parameters

target Input; Pointer to ILE procedure pointer

target is the address of an (16-byte tagged) ILE procedure pointer that addresses the ILE procedure to call.

> **Note**
>
> If the AS/400 PASE program needs additional ILE procedure pointers, it can call an ILE procedure with an argument that is the address of AS/400 PASE memory where the target procedure should store additional ILE pointers.

ILEarglist Input; Pointer to ILE argument list structure

ILEarglist is the address of the structure that contains any argument values to pass to the ILE procedure, as well as memory for a function result returned by the ILE procedure. The ILEarglist argument must point to a buffer that is 16-byte aligned (regardless of whether any of the ILE arguments are MI pointers).

The format and contents of the ILE argument list are specified by the MI architecture. The base structure of an ILE argument list (including the function result area) is specified by type ILEarglist_base. Any argument values for the ILE procedure are stored in memory immediately following the ILEarglist_base type. The specific argument list structure for the ILE procedure identified by the target argument is determined by the list of argument data types addressed by the signature argument.

_ILECALL augments the MI architecture argument list definition to use some of the argument list memory to return a function result from the ILE procedure. Type ILEarglist_base includes a union named result that defines fields to support every ILE function result type handled by _ILECALL.

> **Note**
>
> AS/400 PASE runtime includes two functions that can be used to convert an argument list from the AIX PowerPC ABI used by AS/400 PASE procedures to the MI Architecture form used by ILE procedures: size_ILEarglist and build_ILEarglist.

signature Input; Pointer to a list of arg_type_t values

signature is the address of a list of arg_type_t values that specify the sequence and type of all argument values passed to the ILE procedure. ILE procedures can accept a maximum of 400 arguments. The actual number of ILE arguments processed by the _ILECALL function is determined by the number of entries in the signature list, which is determined by the location of the first ARG_END value in the list. The following values are supported in the signature list:

- **ARG_END**: Specifies the end of the signature list
- **ARG_INT8**: Signed 1-byte integer argument
- **ARG_UINT8**: Unsigned 1-byte integer argument
- **ARG_INT16**: Signed 2-byte integer argument
- **ARG_UINT16**: Unsigned 2-byte integer argument
- **ARG_INT32**: Signed 4-byte integer argument
- **ARG_UINT32**: Unsigned 4-byte integer argument
- **ARG_INT64**: Signed 8-byte integer argument
- **ARG_UINT64**: Unsigned 8-byte integer argument
- **ARG_FLOAT32**: 4-byte floating-point argument
- **ARG_FLOAT64**: 8-byte floating-point argument
- **ARG_MEMPTR**: The argument is an ILE pointer value into which the caller has stored an AS/400 PASE memory address (in member field address). The _ILECALL function converts the AS/400 PASE memory address to an equivalent single-level store address, which it passes to the ILE procedure. The _ILECALL function generally updates the ILE pointer argument value in memory so it contains a tagged MI space pointer (but the memory may not be updated if the target ILE procedure uses ARGOPT linkage).
- **ARG_SPCPTR**: The argument is an ILE pointer value where the AS/400 PASE program has stored a tagged MI Space Pointer.
- **ARG_OPENPTR**: The argument is an ILE pointer value where the AS/400 PASE program has stored an MI pointer of any type (including possibly an untagged/null pointer).

- **Any value in the range 1 to 32767**: The argument is an aggregate (structure or union). The signature list entry value is the length (in bytes) of the aggregate.

Note

An AS/400 PASE program can pass tagged MI Space Pointer arguments to an ILE procedure using either ARG_SPCPTR or ARG_OPENPTR, unless the target ILE procedure uses ARGOPT linkage, in which case ARG_SPCPTR must be used.

`result_type` Input; result_type_t

result_type specifies the type of function result returned by the ILE procedure. The following values are supported:

- **RESULT_VOID (zero)**: No function result, or an aggregate function result
- **RESULT_INT8**: Signed 1-byte integer result
- **RESULT_UINT8**: Unsigned 1-byte integer result
- **RESULT_INT16**: Signed 2-byte integer result
- **RESULT_UINT16**: Unsigned 2-byte integer result
- **RESULT_INT32**: Signed 4-byte integer result
- **RESULT_UINT32**: Unsigned 4-byte integer result
- **RESULT_INT64**: Signed 8-byte integer result
- **RESULT_UINT64**: Unsigned 8-byte integer result
- **RESULT_FLOAT64**: 8-byte floating-point result

  MI architecture specifies that an aggregate function result is returned by an ILE procedure into a buffer that is allocated by the caller and passed to the target procedure using a special field in the argument list. An AS/400 PASE program must set the field result.r_aggregate.addr (in type ILEarglist_base) to the address of the AS/400 PASE memory buffer for the function result before calling an ILE procedure that returns an aggregate result. If the ILE function result is an aggregate, the result_type passed to _ILECALL should be zero.

8.3.7.3 Restrictions

Every module in a *PGM or *SRVPGM object containing a function directly invoked by AS/400 PASE (using _ILECALL) must be teraspace-safe. If any module in the program uses the default of TERASPACE(*NO), AS/400 PASE will not be able to call any function in that program (even a function in a module created as TERASPACE(*YES)).

The _ILECALL function does no character encoding conversions, so the AS/400 PASE program may need to convert argument and result character strings between ASCII/EBCDIC. The AS/400 PASE runtime function iconv can be used for character conversions.

ILE procedure pointers address resources inside an ILE activation group. MI architecture generally prohibits the use of activation group resources from a process other than the owner of the activation group. This restriction means that the child process of a fork generally cannot use ILE procedure pointers inherited from the parent. The child process can, however, use the _ILELOAD function to load a bound program and then use _ILESYM to get an ILE procedure pointer into the (new) activation.

8.3.7.4 Return value

The function result from _ILECALL is zero if the ILE procedure was called and returned normally. The function result is non-zero if errors are found in the signature list or result type. However, most errors detected during the execution of _ILECALL are reported through AS/400 PASE signals.

8.3.7.5 Error conditions

MI exceptions may occur during _ILECALL processing, either because of invalid argument values (such as an invalid MI pointer type value for a ARG_SPCPTR argument) or because of processing done in the target ILE procedure. The Qp2RunPase and Qp2CallPase APIs contain support to convert any MI exception sent to the AS/400 PASE program into an AS/400 PASE signal. Messages are actually sent to the Qp2xxx API invocation because the AS/400 PASE program is not directly addressable as a target invocation for MI operations, such as signaling an exception.

The specific AS/400 PASE signal delivered for an MI exception depends on the OS/400 message identifier for the exception. AS/400 PASE delivers the same signal as ILE C for MCH and CPF message IDs (see *ILE C/400 Programmer's Reference*, SC41-5607, for details), and uses SIGTERM for any exception that is not an MCH or CPF message ID. AS/400 PASE and ILE use different signal numbers, but the AS/400 PASE mapping uses the same

signal condition name (such as SIGSEGV) that is used by ILE C for each message ID.

An AS/400 PASE signal handler can determine whether a signal is associated with an exception message by inspecting the first word (4 bytes) immediately following the struct sigcontext (as defined on AIX) argument passed to the handler. A non-zero value is the message reference key for the OS/400 message that produced the signal. The AS/400 PASE program can pass the message reference key value to an ILE procedure that receives the exception message for more details about the error.

8.3.8 _MEMCPY_WT() and _MEMCPY_WT2() functions

The _MEMCPY_WT and _MEMCPY_WT2 functions copy memory with tags.

Standard memory copy functions, such as memcpy, never produce a usable tagged MI pointer in the target region. _MEMCPY_WT and _MEMCPY_WT2 copy memory in a way that preserves the integrity of any complete (16-byte) tagged pointers copied as long as the source and target have the same alignment with respect to a 16-byte boundary.

_MEMCPY_WT only copies between memory regions that are mapped into the AS/400 PASE address space. _MEMCPY_WT2 can copy between any memory regions addressable through tagged MI space pointers (which do not need to be mapped into the AS/400 PASE address space).

_MEMCPY_WT and _MEMCPY_WT2 are implemented as kernel system calls, so they generally run slower than the memcpy function.

8.3.8.1 Syntax

```
#include <as400_types.h>
 #include <as400_protos.h>

 void*
 _MEMCPY_WT(void             *target,
            const void       *source,
            size_t            length);

 void
 _MEMCPY_WT2(ILEpointer       *target,
             ILEpointer       *source,
             size_t            length);
```

Libraries: Standard C Library (libc.a)

8.3.8.2 Parameters

`target` Input; Pointer to target memory or Pointer to MI space pointer

For the _MEMCPY_WT function, target is the address of the target memory region.

For the _MEMCPY_WT2 function, target is the address of an MI space pointer that addresses the target memory region.

`source` Input; Pointer to source memory or Pointer to MI space pointer

For the _MEMCPY_WT function, source is the address of the source memory region.

For the _MEMCPY_WT2 function, source is the address of an MI space pointer that addresses the source memory region.

`length` Input

length is the number of bytes to copy between the source and target regions.

8.3.8.3 Return value

The function result from _MEMCPY_WT is the address of the source memory region.

_MEMCPY_WT2 returns no function result.

8.3.8.4 Error conditions

Any errors detected by _MEMCPY_WT or _MEMCPY_WT2 are reported as AS/400 PASE signals.

The specified bytes are copied without error if the source and target regions do not have the same alignment with respect to a 16-byte boundary or if only part of a tagged pointer is copied (although the region will not contain a usable tagged pointer).

8.3.9 _SETSPP() function

The _SETSPP function sets a tagged MI space pointer to the teraspace address associated with the specified AS/400 PASE memory address. The _SETSPP function does not produce an error if no memory is mapped at the AS/400 PASE address, and the resulting space pointer always references whatever memory is currently mapped at the teraspace address.

8.3.9.1 Syntax

```
#include <as400_types.h>
#include <as400_protos.h>
void
 _SETSPP(ILEpointer       *target,
         const void       *address);
```

Libraries: Standard C Library (libc.a)

8.3.9.2 Parameters

target Input; Pointer to buffer for MI space pointer

target is the address of an AS/400 PASE memory buffer where the tagged MI space pointer will be returned. The buffer must be 16-byte aligned.

Address Input; Pointer to AS/400 PASE memory

Address is the address of the AS/400 PASE memory that will be addressed by the tagged MI space pointer returned by _SETSPP

Note

A null pointer input value returns a null MI pointer.

8.3.9.3 Return value

_SETSPP returns no function result.

8.3.9.4 Error conditions

Any errors detected by _SETSPP are reported as AS/400 PASE signals (including alignment errors for the input target argument).

8.3.10 _CVTSPP() function

The _CVTSPP function returns the AS/400 PASE address contained in a tagged MI space pointer or zero (NULL) if the space pointer does not address AS/400 PASE memory.

8.3.10.1 Syntax

```
#include <as400_types.h>
#include <as400_protos.h>
void
 _CVTSPP(ILEpointer       *pointer);
```

Libraries: Standard C Library (libc.a)

8.3.10.2 Parameters

pointer Input; Pointer to the MI space pointer

pointer is the address of a tagged MI space pointer. It must be 16-byte aligned.

8.3.10.3 Return value

_CVTSPP returns the AS/400 PASE address of the memory addressed by the MI space pointer or zero (NULL) if the space pointer does not address AS/400 PASE memory.

8.3.10.4 Error conditions

Any errors detected by _CVTSPP are reported as AS/400 PASE signals (including alignment and pointer type errors for the input pointer argument).

8.3.11 _SETCCSID() function

The _SETCCSID function sets the CCSID value assumed by AS/400 PASE runtime and returns the old AS/400 PASE CCSID value. AS/400 PASE runtime uses this CCSID to tag any IFS bytestream file created by an AS/400 PASE program. It also converts selected character data (such as bytestream file names) to and from the encoding required by any OS/400 service used to implement AS/400 PASE runtime support.

8.3.11.1 Syntax

```
#include <as400_types.h>
#include <as400_protos.h>
int
_SETCCSID(int               *ccsid);
```

Libraries: Standard C Library (libc.a)

8.3.11.2 Parameters

ccsid Input; int

ccsid specifies the coded character set identifier (CCSID) to be used by AS/400 PASE runtime. The ccsid value must specify a single-byte encoding (normally an ASCII CCSID) that OS/400 can convert to and from the job default CCSID, or a value of 1208 to indicate that the AS/400 PASE program uses UTF-8 encoding. A special value of -1 can be used to interrogate the current value without changing the AS/400 PASE CCSID.

8.3.11.3 Return value

The function result is either the original AS/400 PASE CCSID, before it was modified, or -1 if an error occurred and the AS/400 PASE CCSID was left unchanged.

8.3.12 systemCL function

The systemCL function runs a CL command, either in the process that invoked the function or in a separate process.

8.3.12.1 Syntax

```
#include <as400_types.h>
#include <as400_protos.h>
int
systemCL(const char     *command,
         int             *flags);
```

Libraries: Standard C Library (libc.a)

8.3.12.2 Authorities and locks

No special authority is required to run the systemCL function. The caller must be authorized to run the specified CL command.

8.3.12.3 Parameters

command Input; Pointer to a null-terminated string

command is the address of a null-terminated character string that contains a CL command with any parameters. The system converts the string from the (ASCII) AS/400 PASE CCSID to the (EBCDIC) job default CCSID before invoking the CL Command Analyzer.

flags Input; Option flags

The flags contain a bitwise *or* of any of the following option values:

- **SYSTEMCL_MSG_STDOUT**: Directs the system to receive OS/400 messages after normal command completion and send them as (ASCII) text lines to `stdout`.
- **SYSTEMCL_MSG_STDERR**: Directs the system to receive OS/400 messages after error command completion and send them as (ASCII) text lines to `stderr`.
- **SYSTEMCL_NOMSGID**: Suppresses message identifiers in text lines written to `stdout` or `stderr` for messages processed on

behalf of SYSTEMCL_MSG_STDOUT and SYSTEMCL_MSG_STDERR. When this option is omitted, message text lines have the form “XXX1234: message text”, where *XXX1234* is the OS/400 message identifier.

- **SYSTEMCL_SPOOL_STDOUT**: Directs the system to process any spooled output files produced by the CL command by writing them as (ASCII) text lines to `stdout`.
- **SYSTEMCL_SPOOL_KEEP**: Directs the system to keep any spooled printer files after they are processed for option SYSTEMCL_SPOOL_STDOUT, instead of the default behavior, which deletes spooled printer files after they are written to `stdout`.
- **SYSTEMCL_FILTER_STDIN, SYSTEMCL_FILTER_STDOUT, SYSTEMCL_FILTER_STDOUT**: These options direct the system to setup filters that convert text between the (ASCII) AS/400 PASE CCSID and (EBCDIC) job default CCSID for any data read or written by the CL command using `stdin`, `stdout`, and `stderr`.

  Processing for these options replaces IFS descriptors 0, 1, or 2 in whatever job runs the CL command with pipes that are handled by ILE pthreads (not AS/400 PASE pthreads) that do CCSID conversion and run in the process that invoked the systemCL function.

  These options only control processing for stream data read and written by the CL command processing program. They do not affect the encoding of text lines written to `stdout` and `stderr` for the SYSTEMCL_MSG_STDOUT, SYSTEMCL_MSG_STDERR, and SYSTEMCL_SPOOL_STDOUT options, which are always converted from the (EBCDIC) job default CCSID to the (ASCII) AS/400 PASE CCSID.
- **SYSTEMCL_SPAWN**: Directs the system to run the command in a separate process that is launched using the ILE spawn function. The OS/400 job that runs the CL command is not multi-thread capable, so it can safely run commands that do not work in a job that is multi-thread capable. If this option is omitted, the CL command is run in the process that invokes the systemCL function.

  Many CL commands are not supported in a multi-threaded

process. Even if the application that calls systemCL is single-threaded, the operation of systemCL may create multiple threads to support options SYSTEMCL_FILTER_STDIN, SYSTEMCL_FILTER_STDOUT, and SYSTEMCL_FILTER_STDERR.

- **SYSTEMCL_SPAWN_JOBLOG**: Directs the system to generate an OS/400 job log for the job submitted using the SYSTEMCL_SPAWN option. A job log may help with problem determination when a command fails.

8.3.12.4 Return value

If the command argument is a null pointer, the function result is zero if system support to invoke the OS/400 Command Analyzer is available or a non-zero value otherwise.

If the SYSTEMCL_SPAWN option is not used, the function result is zero for normal command completion or -1 if an error occurred. An errno value is set if the error is detected by AS/400 PASE runtime, but no errno is set for CL command errors.

If the SYSTEMCL_SPAWN option is used, the function result is the exit code from the spawned job that is returned by the waitpid function, which is non-zero if any error occurred.

8.3.12.5 Error codes

The following error codes can occur when executing the system CL function:

- **EINVAL**: An invalid value was specified for the flags argument.
- **ENOEXEC**: System support to invoke the OS/400 Command Analyzer is not available.

8.3.12.6 SystemCL sample

```
/* sampleCL.c

   example to demonstrate use of sampleCL to execute 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

   Sample execution using QP2SHELL is shown below.
   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]);

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

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

}
```

8.4 Calling AS/400 PASE from ILE

For more information on calling AS/400 PASE from ILE, refer to 6.5, “Starting an AS/400 PASE application” on page 65.

8.4.1 Qp2RunPase information

Refer to 6.5.3, “Invocation from an ILE application: Qp2RunPase” on page 71, for details on the syntax and parameters for Qp2RunPase.

8.4.2 Qp2CallPase information

Refer to 6.5.4, “Invocation from an ILE application: Qp2CallPase” on page 79, for details on the syntax and parameters for Qp2CallPase.

Part 3. AS/400 PASE from an AS/400 perspective

This part is intended primarily for AS/400 sites that are trying to use a ported application. You'll also find this part to be helpful if you are trying to port and test your application on an AS/400 system.

Chapter 9. UNIX architecture from an AS/400 perspective

The UNIX architecture can generally be portrayed as a series of concentric circles. The innermost circle is the hardware being supported. Outside of that is the operating system kernel. Next are the libraries directly supported by the UNIX operating system. Next are the shells and utilities provided by the operating system. After that come libraries and other middleware provided by the application developers. On the outermost circle are the UNIX applications. Figure 13 shows how this looks to a UNIX developer.

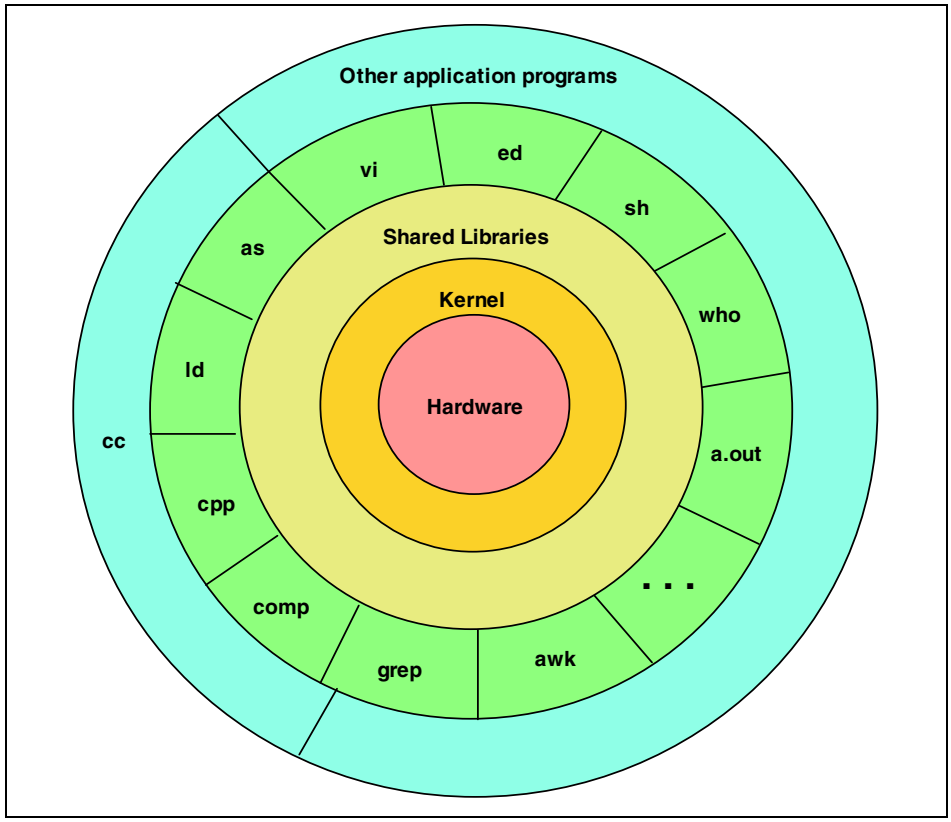


Figure 13. UNIX architecture

Figure 14 on page 146 shows that AS/400 PASE is an environment on the AS/400 system designed to behave like AIX to the applications. Instead of mapping most of the services directly to the hardware, they are mapped into OS/400 services. At runtime, the system operator can view AS/400 PASE from the lower half of the circle. Because all of the services are mapped into

OS/400 services, the applications can be managed just like any other AS/400 application.

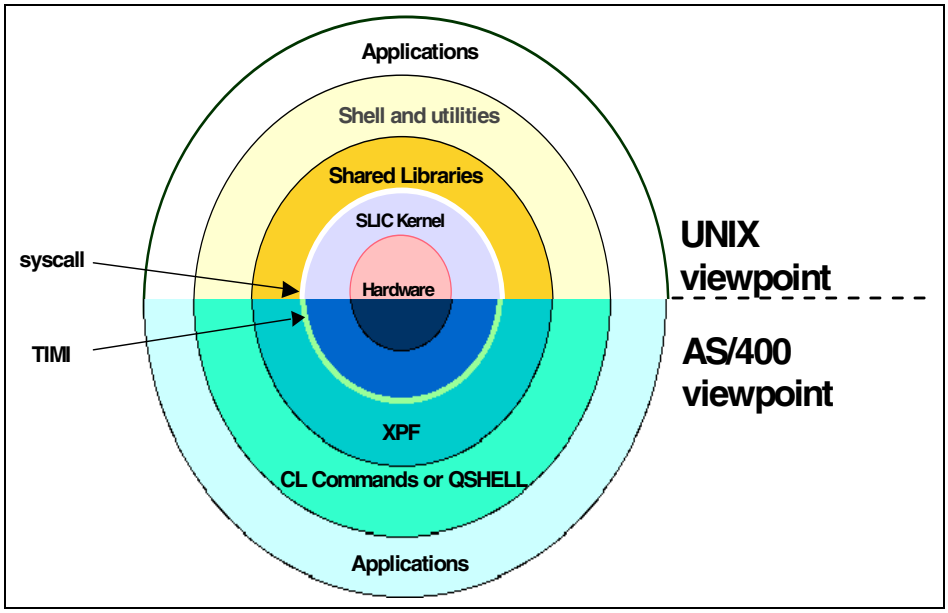


Figure 14. AS/400 PASE in UNIX terms

Chapter 10. AS/400 PASE porting examples

We used the following criteria when choosing an application to port:

- Source or binaries freely available for download from the Internet
- Showcases the support of the X Windows library support in AS/400 PASE
- Identifies some of the more simple issues that may be encountered

Searching the World Wide Web, we found the Open Source Project based on IBM's Visualization Data Explorer, known as OpenDX, located at: `http://www.opendx.org`

OpenDX is a software package for the visualization of scientific, engineering, and analytical data. OpenDX requires no porting since the binaries are available and work without source changes, except to one shell script.

One of the early problems discovered is that many open source sites distribute archives that were generated using the GNU zip utility. Obtaining and porting the GNU zip is the first task in the porting examples.

The final porting example is GNU perl. Perl is a popular and powerful scripting tool that is used as a base in many Web and e-mail sites. The popular midrange mailing list, Midrange-L, is implemented using Perl. It can be accessed online at: `http://www.midrange.com`

Since these examples merely show the techniques of compilation or binary extractions, a more thorough examination of an application would include an API analysis. For the OpenDX example, this process was followed and the results are shown in 5.2.1, "API analysis example" on page 46.

These examples use the C shell syntax. Use the appropriate syntax for the shell being used.

10.1 GNU zip

Many open source sites on the Internet distribute archives that were generated using the GNU zip utility. Since this utility is not part of AS/400 PASE, we provide this utility here as our first porting example. Perform the following steps and the utility will be generated and can be used to un-zip many other archives found on the Web.

Starting with AIX

Perform the following steps on AIX.

1. From your AIX workstation, open a browser to the URL:
   `ftp://ftp.gnu.org/gnu/gzip/`

   Note

   The GNU FTP site is a very busy site. You may not succeed with the login process. Either try again later, or you may go to the GNU Web site, `http://www.gnu.org`, and proceed to download from a mirrored site, that is, a site that copies the GNU distributions.

2. Select **gzip-1.2.4a.tar** or another version that is desired (Figure 15).

   Note

   Select a version that has a suffix of .tar, and not .gz. A suffix ending in .gz is a GNU zipped tar file. The point of this example implies that the utility is not available in the first place. Otherwise, you can skip this exercise.

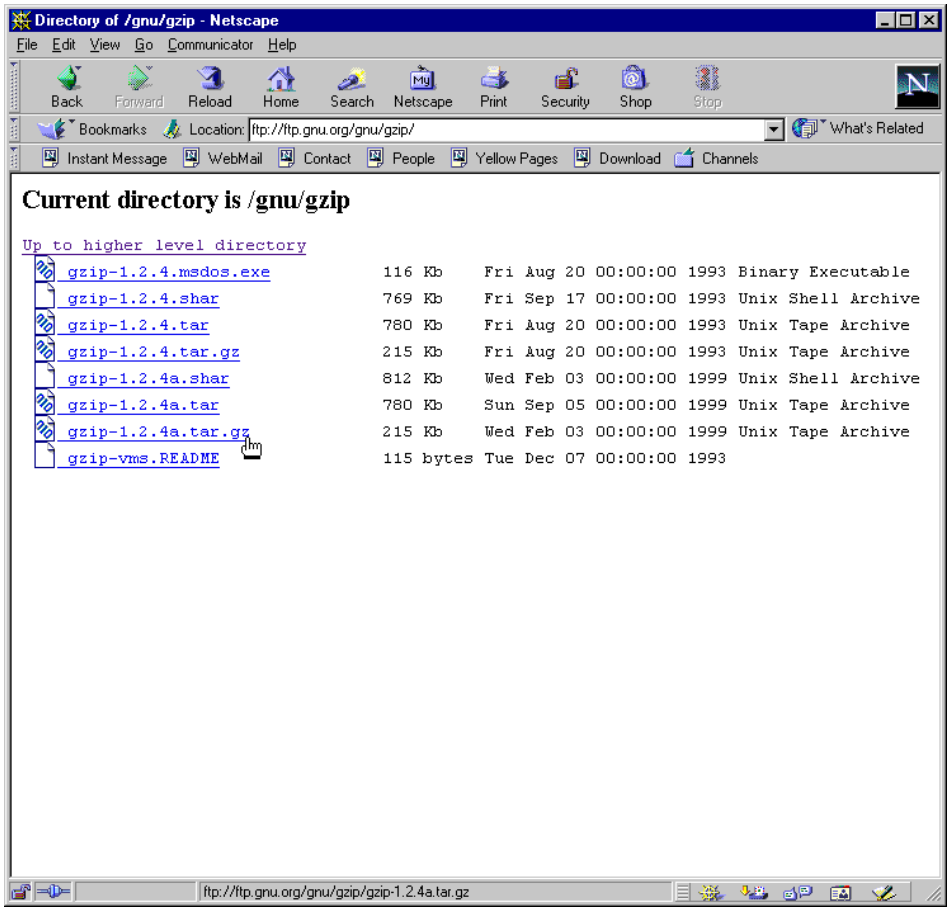


Figure 15. The GNU gzip FTP directory

3. Press Shift and click to select a file to download. Save the file to a desired directory.
4. From an AIX terminal window, change the directory to the saved location of the tar file.
5. Untar the archive: `tar -xvf gzip-1.2.4a.tar`
6. Change the directory to the created directory:

   `cd gzip-1.2.4a`
7. Review the README and INSTALL documentation files for background information.

8. If running in C shell, enter:

   ```
   setenv CC xlc
   ```

   If in Bourne or Korn shell, enter:

   ```
   export CC=xlc
   ```

9. Run the configuration script: `./configure`
10. Run the make utility to compile the code: `make`

    After completing these steps, a gzip and a gunzip utility are created. Actually, the gunzip is a symbolic link to the gzip program. Other utilities also were created, but they are not used in any of the following examples. These tools, gzip and gunzip, can be used solely on the AIX machine, or the unzipping process can be done in the AS/400 PASE environment as well.

    To perform the gunzip process with AS/400 PASE, FTP the gzip binary file to the AS/400 system with AS/400 PASE into the IFS file system, in binary mode, to a directory of your choice. The /usr/local/bin directory may be a preferable choice and may need to be created.

11. Enter the following statement, where *pase400* is the name of the AS/400 PASE system:

    ```
    ftp pase400
    ```

12. Enter a valid user ID and password when prompted to do so.
13. Set to binary mode transfer by entering:

    ```
    bin
    ```

14. Make the target directory by entering:

    ```
    mkdir /usr
    mkdir /usr/local
    mkdir /usr/local/bin
    ```

15. Send the file to the AS/400 file system by entering:

    ```
    put gzip /usr/local/bin/gzip
    ```

16. Exit FTP by entering:

    ```
    quit
    ```

10.2 OpenDX

The next step is to acquire the OpenDX binaries from the site:
`http://www.opendx.org`

Starting with AIX

Perform the following steps on AIX.

1. From your AIX machine, open a browser to the Web site:
   `http://www.opendx.org`
2. Click the **Continue** button.
3. Select **Downloads** on the left side of the Web page, as shown in Figure 16.

Figure 16. OpendDX home page

4. Click **download** in parentheses after “Binary distributions”.

5. Fill out the questionnaire, and click the **Agree** button.
6. Select **OpenDx version 4.1.0**, which is the OpenDX version for AIX 4.1.

   Figure 17 shows the Web page from OpenDX with all the binaries available. Select **AIX v4.1 or higher** for download.

> **Note**
>
> OpenDX binaries for AIX Version 4.0.6 were found to work unacceptably. Download the indicated version or newer for a better demonstration.

Figure 17. Selecting the AIX OpenDX binary tar file

7. Save the GNU zipped (suffix of.gz) file to a suitable directory.
8. Download the samples (**dxsamples-4.0.8.tar.gz**) under the “Common files” heading.
9. FTP the two downloads to the AS/400 PASE system:
   a. Enter: `ftp paseas400`
   b. Set binary mode transfer: `bin`
   c. Enter:

      ```
      put opendx-4.1.0-aix41.tar.gz
      /home/spartz/opendx/opendx-4.1.0-aix41.tar.gz
      ```

   d. Enter:

      ```
      put dxsamples-4.0.8.tar.gz
      /home/spartz/opendx/dxsamples-4.0.8.tar.gz
      ```

Switching to the AS/400 system

Complete the following steps on the AS/400 system.

10. Sign on to the AS/400 system with a 5250 emulation program or at the system console.
11. Enter the AS/400 PASE terminal shell by entering the following statement on the command line prompt:

    ```
    call qp2term
    ```

12. Create a symbolic link for gunzip by entering:

    ```
    ln -s /usr/local/bin/gzip /usr/local/bin/gunzip
    ```

13. Change the directory to where the tar files were sent by FTP by entering:

    ```
    cd /home/spartz/opendx
    ```

14. Unzip the zipped tar files by entering:

    ```
    /usr/local/bin/gunzip opendx-4.1.0-aix41.tar.gz
    /usr/local/bin/gunzip dxsamples-4.0.8.tar.gz
    ```

    Figure 18 on page 154 shows the command to unzip the OpenDX tar file.

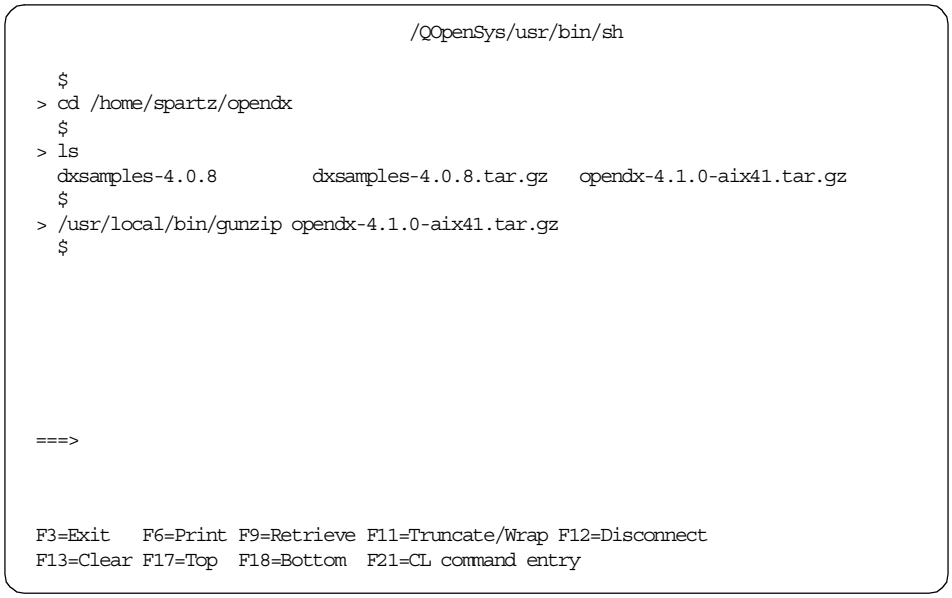


Figure 18. Unzipping the OpendDX tar file

15. Change the directories to the root (/). Enter:

    ```
    cd /
    ```

16. Untar OpenDX by entering:

    ```
    tar -xvf /home/spartz/opendx/opendx-4.1.0-aix41.tar
    ```

    At this point, the normal process would be to have the binaries analyzed for AS/400 PASE compatibility. This did not deter us from trying it first. The analysis was performed and is shown in 5.2.1, “API analysis example” on page 46.

17. Change the directories to OpenDX by entering:

    ```
    cd /usr/local/dx
    ```

18. Untar the OpenDX samples tar file by entering:

    ```
    tar -xvf /home/spartz/opendx/dxsamples-4.0.8.tar
    ```

19. Create a symbolic link for “samples”, since the default shell script uses that name by default. Enter:

    ```
    ln -s dxsamples-4.0.8 samples
    ```

20. Ensure that the file /dev/null exists. The OpenDX startup script requires its presence. Enter:

    ```
    touch /dev/null
    ```

The basic setup is complete. There is, however, one shell script change that is required to run OpenDX under AS/400 PASE. The /usr/local/dx/bin/dxworker shell script uses the system utility `uname` to make platform specific decisions. On AS/400 PASE, the output value is “OS400”. Figure 19 shows the `grep` utility locating the script case statement of the dxworker script where the AIX case statement needs to be changed to OS400.

```
                                /QOpenSys/usr/bin/sh

  $
> cd /usr/local/dx/bin
  $
> ls
  dx                dxworker           dxworker.original  mdf2c
  $
> grep AIX dxworker
    case AIX*:
      # and .net file reading on AIX.
      # this is for AIX 3.2.3e, so we get the right libs at run time
        echo "WARNING!  AIX Version seems to be other than 3.2 or 4.x"
  $



===>



F3=Exit   F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
F13=Clear F17=Top  F18=Bottom  F21=CL command entry
```

Figure 19. Locating shell script change

This can also be changed within the AS/400 PASE environment without editing the file remotely or sending the file via FTP to another system to edit, and transferring it back.

21. Change to the dx directory by entering:

    ```
    cd /usr/local/dx/bin
    ```

22. Make a backup copy of the shell script by entering:

    ```
    cp dxworker dxworker.original
    ```

23. Change “AIX*:” to “OS400:”. Enter:

    ```
    sed “s/AIX.:/OS400:/” dxworker.original > dxworker
    ```

An alternative to using the AS/400 PASE `sed` utility, is to use the OS/400 `EDTF` CL command, entered as follows to change the script file. Enter this on the AS/400 terminal screen to invoke the editor:

```
EDTF '/usr/local/dx/bin/dxworker'
```

24. The OpenDX shell scripts look for some binaries in /bin. This directory is not automatically created for AS/400 PASE. See 6.4, “Configuration tips” on page 62. Create a symbolic link by entering:

```
ln -s /QopenSys/usr/bin /bin
```

Before the application is launched, you must identify the X server that is to render the displayed output. Enter the following command in your QP2TERM session:

```
export DISPLAY=10.5.118.190:0
```

This command exports the environment variable DISPLAY to the running shell with the given value. X client library support uses this environment variable to determine the destination host of the rendered screen window. The value of the variable can either be an IP address of the TCP host running the X server for displaying the graphics or a hostname that can be resolved to an IP address. The “:0” identifies the display number to which the X client application connects. This is correct under most circumstances.

You must also ensure that the host acting as the X server is authorized to accept the connection from the client application, OpenDX. On a UNIX workstation, the `xhost +hostname` command authorizes *hostname* to connect to this X server. On a Microsoft Windows platform, a product, such as Hummingbird Exceed or XWin32, needs to be installed. The Xconfig tool in Exceed enables the Security user interface to edit the xhosts.txt, placing the name of the hosts authorized to connect to the Exceed X server. The default behavior of XWin32 is to allow any client.

Add /usr/local/bin, or the target directory OpenDX that was installed, to your path environment variable, such as:

```
export PATH=/usr/local/bin:$PATH
```

If /usr/local/bin was used, launch OpenDX with:

```
dx &
```

Otherwise, use the script switch -dxroot to specify the path to the OpenDX root:

```
dx -dxroot /path &
```

Environment variables can also override the script defaults. See shell script dxworker, untar-ed in usr/local/dx/bin.

The startup windows appear on the workstation desktop, as shown in Figure 20.

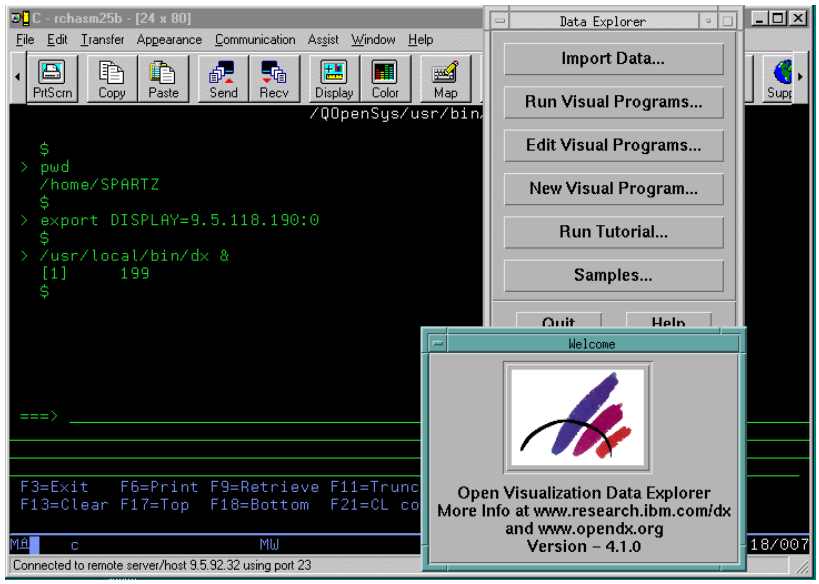


Figure 20. Running OpenDX in AS/400 PASE

One of the demo samples is of a water molecule. To run it, follow these steps:

1. Click **Edit Visual Programs** from the OpenDX user interface panel.
2. From the Net File Selection dialog, in the Selection input field, enter:

   `/usr/local/dx/samples/tutorial/example1.net`
3. Click **OK**. In a few moments, several new panels will open.
4. In the Visual Program Editor, under the Execute pull-down, select **Execute Once**. The graphic image of the water molecule shown in Figure 21 on page 158 appears.

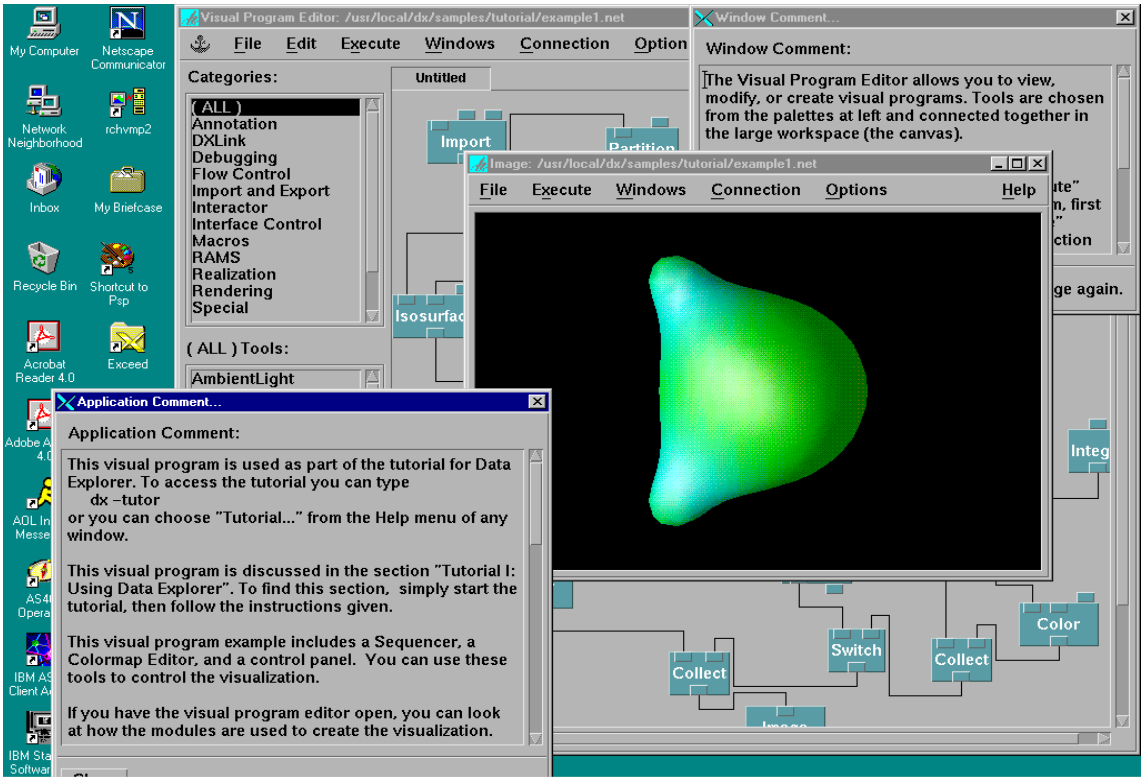


Figure 21. OpenDX tutorial exampl1.net

In the Image window, under the Options pull-down menu, View Control provides a method of changing the image, such as rotating it and so on.

As an alternative to using QP2TERM to launch OpenDX, a QP2SHELL command can be used:

```
CALL PGM(QP2SHELL) PARM('/QOpenSys/bin/sh' 'export
DISPLAY=10.5.118.190:0;export PATH=/usr/local/bin:$PATH;dx &')
```

Or, setup the environment variables in OS/400 before invoking QP2SHELL. If the variable already exists, use the `WRKENVVAR` command and edit the variable as needed. Add a prefix to environment variables with the prefix string ‘PASE_’ to ensure the variable is unique to AS/400 PASE. See 6.5.1, “Invocation from a 5250 terminal screen: QP2SHELL” on page 65, for further information:

```
QSYS/ADDENVVAR ENVVAR(PASE_PATH)
VALUE('/usr/local/bin:/QOpenSys/usr/bin:/usr/ccs/bin:/usr/sbin:.:/usr/bin'
)
```

```
QSYS/ADDENVVAR ENVVAR(PASE_DISPLAY) VALUE('10.5.118.190:0')
CALL PGM(QP2SHELL) PARM('/QOpenSys/bin/sh' 'dx &')
```

10.3 GNU perl

As another example of adding Open Source tools to an AS/400 system using AS/400 PASE, the GNU perl script interpreter is ported.

Note: All of the steps are listed in this example, including the steps that are redone as a typical part of a new port. If you want to build perl, be sure to read the entire example first before trying it yourself.

Starting with AIX

Perform the following steps on AIX.

1. From your AIX machine, open a browser to the URL:
   `ftp://ftp.gnu.org/gnu/perl/`
2. Select **perl-5.005.03.tar.gz** or another version that is desired for download to an AIX workstation.
3. Unzip the zipped tar file by entering:

   ```
   /usr/local/bin/gunzip perl perl-5.005.03.tar.gz
   ```
4. Untar the tarball. Enter:

   ```
   tar -xvf perl-5.005.03.tar
   ```
5. Change the directory to the perl directory. Enter:

   ```
   cd perl-5.005.03
   ```
6. Set the environment variable for the compiler (C shell syntax) by entering:

   ```
   setenv CC xlc
   ```

   Note: The following steps show the normal porting process. This example shows that a compile is successful, but the executable fails.
7. Configure the build by entering:

   ```
   ./Configure -d
   ```
8. Press the Enter key when prompted to do so.
9. Build the interpreter. Enter:

   ```
   make
   ```
10. Build the test scripts. Enter:

    ```
    make test
    ```

11. Create a tarball of the test directory, lib directory, and Configure file for a full test of the compiled perl on AS/400 PASE. To do so, enter:

```
tar -cvf ../perltest.tar t lib Configure
```

12. FTP the tarball to the AS/400 PASE system. In this example, the /home/spartz/perl5 directory is used:

```
mkdir /home/spartz/perl5
```

13. FTP the perl binary, in binary mode, to the AS/400 PASE system. Enter:

```
bin
put perl /home/spartz/perl5
```

14. FTP an example perl script to the AS/400 PASE system. Enter:

```
put id.pl /home/spartz/perl5
```

The contents of id.pl are listed here:

```
#!/usr/bin/perl
# File id.pl created by Daryl Spartz at 15:19:09 on Tue Apr 11 2000.

#if ($#ARGV < 0 || $ARGV[0] eq '-h') {
#    print "Usage:  id [-h]\n";
#    exit;
#}

sub u { local($name) = getpwuid($_[0]); $name && "($name)";}
sub g { local($name) = getgrgid($_[0]); $name && "($name)";}
sub bynum { $a <=> $b; }

print "uid=$<", &u($<);
print " gid=", $(+0, &g($();
print " euid=$>", &u($>) if $> != $<;
print " egid=", $)+0, &g($)) if $) != $(;
@groups=split(' ', $(); shift(@groups);
@groups && print " groups=",
    join(',', sort bynum grep(($_ .= &g($_)) || 1, @groups));
print "\n";
```

Switching to the AS/400 system

Complete the following steps on the AS/400 system.

15. Sign on to the AS/400 system, and invoke a QP2TERM session. Enter:

```
call qp2term
```

16. Untar the perl test suite tarball:

```
cd /home/spartz/perl5
tar -xvf ../perltest.tar
```

17. Change the directory to where perl and the example script exist. Then, invoke perl. The results are shown in Figure 22.

```
                                /QOpenSys/usr/bin/sh

  $
> cd /home/spartz/perl5
> ls
  Configure  id.pl      lib        perl       t
> ./perl id.pl
  Could not load program ./perl:
          Dependent module libnsl.a(shr.o) could not be loaded.
  Could not load module libnsl.a(shr.o).
  Error was: No such file or directory
  Could not load module ./perl.
  $



===>



F3=Exit   F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
F13=Clear F17=Top  F18=Bottom  F21=CL command entry
```

Figure 22. First test of ported perl

This problem occurs because the AS/400 PASE set of supplied libraries does not include libnsl.a. FTPing the necessary library or libraries is not a valid option. See 3.2, “Licensing issues” on page 33. What should you do now? In this case, see if the library is actually used and, if not, remove the dependency. For perl, the failing library is not used, so we can remove it. The remaining steps correct the problem and rebuild the program.

Switching to AIX

Perform the following steps on AIX.

18. Make a backup copy of the configuration file by entering:

```
cp ./Configure ./Configure.original
```

19. Make the file editable by entering:

```
chmod +w ./Configure
```

20. Edit the file by removing nsl from the library list. Enter:

```
vi ./Configure
```

21. Locate the library within the editor. Enter:

```
/nsl
```

22. Delete the library name. Press the letter “x” four times to delete each character of file name.

23. Save and quit the editor by entering:

```
:wq!
```

24. Cleanup from the previous compile. Enter:

```
make realclean
rm -f config.sh
```

25. Repeat step 7 on page 159 through 13 on page 160.

Switching to the AS/400 system

Complete the following steps on the AS/400 system.

26. Launch perl again. The new results are shown in Figure 23.

```
                                /QOpenSys/usr/bin/sh

  $
> cd /home/spartz/perl5
  $
> ls
  Configure  id.pl      lib        perl       t
  $
> perl id.pl
  uid=603(SPARTZ) gid=0
  $




===>



F3=Exit   F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
F13=Clear F17=Top  F18=Bottom  F21=CL command entry
```

Figure 23. Rebuild perl running example

The basic mechanism of perl appears to work. However, when running the test suite, additional problems appear.

27. Invoke the perl test suite by entering:

```
cd /home/spartz/perl5/t
perl TEST
```

At this point, examination of the source code is necessary. Using `grep` to locate the error message string for the glob failures, `grep "child exit" *.c`, the module pp_hot.c is identified as the source of the message. The error is discovered to be the path to the C shell. The code uses a string point to the C shell as /usr/bin/csh. In AS/400 PASE, the correct path is /QOpenSys/usr/bin/csh. This and the path to the `sed` utility are not correct. Another way to "discover" these errors is to use the `string` utility, as in `string perl | grep bin`. The output of this command shows the incorrect paths to `csh` and to `sed`.

To fix these errors, the following steps are needed:

Switching to AIX

Perform the following steps on AIX.

28. Cleanup the compiled environment by entering:

```
make realclean
```

29. Rebuild the configuration. Enter:

```
./Configure -d
```

Note

This time, *do not* press Enter at the prompt to edit config.sh.

a. Edit config.sh:

```
!vi config.sh
```

b. Locate strings:

```
/full_
```

c. Change the path to csh and sed to /QOpenSys/usr/bin. See 6.4, "Configuration tips" on page 62, for a commentary on problems with /usr/bin.

   i. Move the cursor to the starting double quote (").
   ii. Press the "a" key for insert after.
   iii. Type `/QOpenSys` on the lines with *full_csh* and *full_sed* in them
   iv. Press the ESC key to exit the entry mode.
   v. File your changes by entering:

```
:wq!
```

d. Press Enter to continue.

30. Remake the program. Enter:

```
make
```

31. Remake the tests:

```
make test
```

> **Note**
>
> This time the test indicates errors on AIX. This is because AIX does not have a /QOpenSys/usr/bin path. This can be worked around by switching to root authority and creating a symbolic link on AIX from /usr/bin to /QOpenSys/usr/bin: ln -s /usr/bin /QOpenSys/usr/bin.

32. FTP the new perl binary to the AS/400 PASE system and rerun the test suite. Refer to step 27.

> **Switching to the AS/400 system**
>
> Complete the following steps on the AS/400 system.

The following text contains output generated to the QP2TERM session as a result of running the perl tests. Out of 190 tests, 187 were successful. The three failures are due to restrictions within AS/400 PASE, such as no tty support.

```
> ls
   opendx  perl5
   $
 > cd perl5
   $
 > cd t
   $
 > perl TEST
   base/cond.....ok
   base/if.....ok
   base/lex.....ok
   base/pat.....ok
   base/rs.....ok
   base/term.....ok
   comp/cmdopt.....ok
   comp/colon.....ok
   comp/cpp.....skipping test on this platform
   comp/decl.....ok
   comp/multiline.....ok
   comp/package.....ok
   comp/proto.....ok
   comp/redef.....ok
   comp/require.....ok
   comp/script.....ok
   comp/term.....ok
   comp/use.....ok
   cmd/elsif.....ok
```

```
cmd/for............ok
cmd/mod............ok
cmd/subval.........ok
cmd/switch.........ok
cmd/while..........ok
io/argv............ok
io/dup.............ok
io/fs..............ok
io/inplace.........ok
io/iprefix.........ok
io/pipe............ok
io/print...........ok
io/read............ok
io/tell............ok
op/append..........ok
op/arith...........ok
op/array...........ok
op/assignwarn......ok
op/auto............ok
op/avhv............ok
op/bop.............ok
op/chop............ok
op/closure.........ok
op/cmp.............ok
op/cond............ok
op/context.........ok
op/defins..........ok
op/delete..........ok
op/die.............ok
op/die_exit........ok
op/do..............ok
op/each............ok
op/eval............ok
op/exec............ok
op/exp.............ok
op/flip............ok
op/fork............ok
op/glob............ok
op/goto............ok
op/goto_xs.........ok
op/grep............ok
op/groups..........skipping test on this platform
op/gv..............ok
op/hashwarn........ok
op/inc.............ok
op/index...........ok
op/int.............ok
op/join............ok
op/list............ok
op/local...........ok
op/magic...........ok
op/method..........ok
op/misc............ok
op/mkdir...........ok
op/my..............ok
op/nothread........ok
op/oct.............ok
op/ord.............ok
op/pack............ok
op/pat.............ok
op/pos.............ok
op/push............ok
op/quotemeta.......ok
```

```
op/rand............ok
op/range...........ok
op/read............ok
op/readdir.........ok
op/recurse.........ok
op/ref.............ok
op/regexp..........ok
op/regexp_noamp....ok
op/repeat..........ok
op/runlevel........ok
op/sleep...........ok
op/sort............ok
op/splice..........ok
op/split...........ok
op/sprintf.........ok
op/stat............Can't open /dev/tty--run t/TEST outside of make.
FAILED at test 35
op/study...........ok
op/subst...........ok
op/substr..........ok
op/sysio...........ok
op/taint...........getpwent: No such file or directory
FAILED at test 142
op/tie.............ok
op/tiearray........ok
op/tiehandle.......ok
op/time............ok
op/tr..............ok
op/undef...........ok
op/universal.......ok
op/unshift.........ok
op/vec.............ok
op/wantarray.......ok
op/write...........ok
pragma/constant....ok
pragma/locale......ok
pragma/overload....ok
pragma/strict......ok
pragma/subs........ok
pragma/warning.....ok
lib/abbrev.........ok
lib/anydbm.........ok
lib/autoloader.....ok
lib/basename.......ok
lib/bigint.........ok
lib/bigintpm.......ok
lib/cgi-form.......ok
lib/cgi-function...ok
lib/cgi-html.......ok
lib/cgi-request....ok
lib/checktree......ok
lib/complex........ok
lib/db-btree.......skipping test on this platform
lib/db-hash........skipping test on this platform
lib/db-recno.......skipping test on this platform
lib/dirhand........ok
lib/dosglob........ok
lib/dumper-ovl.....ok
lib/dumper.........ok
lib/english........ok
lib/env............ok
lib/errno..........ok
lib/fatal..........ok
```

```
lib/fields.........ok
lib/filecache......ok
lib/filecopy.......ok
lib/filefind.......ok
lib/filehand.......ok
lib/filepath.......ok
lib/filespec.......ok
lib/findbin........ok
lib/gdbm...........skipping test on this platform
lib/getopt.........ok
lib/h2ph...........ok
lib/hostname.......ok
lib/io_dup.........ok
lib/io_pipe........ok
lib/io_sel.........ok
lib/io_sock........ok
lib/io_taint.......ok
lib/io_tell........ok
lib/io_udp.........ok
lib/io_xs..........ok
lib/ipc_sysv.......ok
lib/ndbm...........ok
lib/odbm...........Can't load '../lib/auto/ODBM_File/ODBM_File.so' for module
 ODBM_File: dlopen: ../lib/auto/ODBM_File/ODBM_File.so: No such file or direc
tory at ../lib/DynaLoader.pm line 169.

 at lib/odbm.t line 15
FAILED at test 0
lib/opcode.........ok
lib/open2..........ok
lib/open3..........ok
lib/ops............ok
lib/parsewords.....ok
lib/ph.............ok
lib/posix..........ok
lib/safe1..........ok
lib/safe2..........ok
lib/sdbm...........ok
lib/searchdict.....ok
lib/selectsaver....ok
lib/socket.........ok
lib/soundex........ok
lib/symbol.........ok
lib/textfill.......ok
lib/texttabs.......ok
lib/textwrap.......ok
lib/thread.........skipping test on this platform
lib/tie-push.......ok
lib/tie-stdarray...ok
lib/tie-stdpush....ok
lib/timelocal......ok
lib/trig...........ok
Failed 3 test scripts out of 190, 94.74% okay.
   ### Since not all tests were successful, you may want to run some
   ### of them individually and examine any diagnostic messages they
   ### produce.  See the INSTALL document's section on "make test".
   ### If you are testing the compiler, then ignore this message
   ### and run
   ###      ./perl harness
   ### in the directory ./t.
   ###
   ### Since most tests were successful, you have a good chance to
   ### get information with better granularity by running
```

```
   ###     ./perl harness
   ### in directory ./t.
u=0.37  s=1.12  cu=46.96  cs=117.42  scripts=182  tests=6483
$
```

Chapter 11. Work management

This chapter illustrates how AS/400 PASE programs fit into the AS/400 system's model of job control.

11.1 Viewing AS/400 PASE programs running on the AS/400 system

The normal work management functions can be used with AS/400 PASE jobs. In this series of examples, AUSER signs on to the AS/400 system and starts the AS/400 PASE environment. They then run OpenDX, which is the sample application that we ported in 10.2, "OpenDX" on page 150.

Figure 24 shows AUSER on the WRKACTJOB command display, before starting the AS/400 PASE environment.

```
                        Work with Active Jobs                    AS25B
                                                              05/03/00  13:04:14
CPU %:     .1     Elapsed time:   00:08:41     Active jobs:   190

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
     QBATCH         QSYS        SBS      .0                 DEQW
     QCMN           QSYS        SBS      .0                 DEQW
     QCTL           QSYS        SBS      .0                 DEQW
     QHTTPSVR       QSYS        SBS      .0                 DEQW
     QINTER         QSYS        SBS      .0                 DEQW
       ADEVICE      AUSER       INT      .0  MNU-MAIN       DSPW
       QPADEV0001   JCOOK       INT      .0  PGM-BUPMENUNE  DSPW
       QPADEV0006   COOK        INT      .0  PGM-BUPMENUNE  DSPW
       QPADEV0009   KRUEGER     INT      .0  CMD-WRKACTJOB  RUN
                                                                More...
Parameters or command
===>
F3=Exit   F5=Refresh       F7=Find      F10=Restart statistics
F11=Display elapsed data   F12=Cancel   F23=More options   F24=More keys
```

Figure 24. WRKACTJOB with AUSER on the main menu

Figure 25 on page 170 shows the WRKACTJOB display after AUSER entered `CALL QP2TERM` on the OS/400 command line to start the AS/400 PASE environment. Notice that the user is now running a program named QP2TERM, and it is in a DEQW status.

Figure 26 on page 171 shows AUSER's terminal after starting the OpenDX application. AUSER has been running Example1 and has several X windows

open on their PC, that are each supported by individual forked jobs in the AS/400 PASE environment. AUSER entered the `ps -ef` command to see all of the active AS/400 PASE processes.

AS/400 PASE processes are identified with numeric process IDs. There is a qshell command named `getjobid` that can be used to correlate between the process IDs and AS/400 job names. QSHELL commands aren't directly supported in QP2Shell, so enter the following line to find out what the process name is:

```
system qsh "cmd('getjobid 597')"
```

Figure 27 and Figure 28 on page 172 show two different views of the WRKACTJOB panel with all of the AS/400 PASE jobs displayed. Notice that they look just like other jobs on the system. See if you can determine which job matches process ID 597 from the AS/400 PASE environment.

```
                          Work with Active Jobs                    AS25B
                                                                05/03/00  13:18:56
CPU %:     .2     Elapsed time:   00:23:23     Active jobs:   192

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
     QBATCH         QSYS        SBS      .0                 DEQW
     QCMN           QSYS        SBS      .0                 DEQW
     QCTL           QSYS        SBS      .0                 DEQW
     QHTTPSVR       QSYS        SBS      .0                 DEQW
     QINTER         QSYS        SBS      .0                 DEQW
       ADEVICE      AUSER       INT      .0  PGM-QP2TERM    DEQW
       QPADEV0001   JCOOK       INT      .0  PGM-BUPMENUNE  DSPW
       QPADEV0003   SPARTZ      INT      .0  PGM-QCMD        DSPW
       QPADEV0006   COOK        INT      .0  PGM-BUPMENUNE  DSPW
                                                                More...
Parameters or command
===>
F3=Exit   F5=Refresh       F7=Find      F10=Restart statistics
F11=Display elapsed data   F12=Cancel   F23=More options   F24=More keys
```

Figure 25. WRKACTJOB with AUSER running AS/400 PASE

```
                                /QOpenSys/usr/bin/sh

  $
 > dx &
  [1]    597
  $
 > ps -ef
     UID PID PPID   C   STIME   TTY  TIME CMD
    AUSER 596 362   0 15:12:33    -  0:00 /QOpenSys/usr/bin/sh -i
  AUSER 597 596   0 15:13:05   - 0:00 /usr/local/dx/bin_ibm6000/startupui
      AUSER  610 597    0  15:14:10      -   0:02  /usr/local/dx/bin_ibm6000/dxui
-processors 1
      AUSER  623 610    0  15:14:20    -   0:00  csh -f /usr/local/dx/bin/dxworker -exonly
                                          -local -directory / -processors 1
      AUSER  640 623    0  15:14:27      -   0:02  /usr/local/dx/bin_ibm6000/dxexec -r -p1
-B
    AUSER 642 596   0 15:15:11    - 0:00 ps -ef
  $
 > system qsh "cmd('getjobid 597')"
  Process identifier 597 is 075611/AUSER/QP2FORK
  $

 ===>

 F3=Exit  F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
 F13=Clear  F17=Top   F18=Bottom   F21=CL command entry
```

Figure 26. AS/400 PASE environment running OpenDX and looking at jobs

```
                          Work with Active Jobs                      AS25B
                                                                  05/03/00  15:16:50
 CPU %:     .6     Elapsed time:   02:21:18     Active jobs:   197

 Type options, press Enter.
   2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
   8=Work with spooled files   13=Disconnect ...

 Opt  Subsystem/Job  User        Number  Type  CPU %  Threads
        QPADEV0006   COOK        075037  INT      .0        1
        QPADEV0009   KRUEGER     075355  INT      .0        1
        QP2FORK      AUSER       075611  BCI      .0        1
        QP2FORK      AUSER       075624  BCI      .0        1
        QP2FORK      AUSER       075637  BCI      .0        1
        QP2FORK      AUSER       075654  BCI      .0        1
        QP2SHELL     AUSER       075610  BCI      .0        1
      QSERVER        QSYS        074847  SBS      .0        1
        QPWFSERVSD   QUSER       074898  BCH      .0        1
                                                                        More...
 Parameters or command
 ===>
 F3=Exit    F5=Refresh    F7=Find    F10=Restart statistics    F11=Display status
 F12=Cancel    F17=Top    F18=Bottom    F23=More options      F24=More keys
```

Figure 27. WRKACTJOB showing the AS/400 PASE jobs

```
                        Work with Active Jobs                   AS25B
                                                             05/03/00  15:19:12
CPU %:     .6     Elapsed time:   02:23:40     Active jobs:   197

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
       QPADEV0006   COOK        INT      .0  PGM-BUPMENUNE   DSPW
       QPADEV0009   KRUEGER     INT      .0  CMD-WRKACTJOB   RUN
       QP2FORK      AUSER       BCI      .0                  SELW
       QP2FORK      AUSER       BCI      .0                  SELW
       QP2FORK      AUSER       BCI      .0                  THDW
       QP2FORK      AUSER       BCI      .0                  SELW
       QP2SHELL     AUSER       BCI      .0                  TIMW
     QSERVER        QSYS        SBS      .0                  DEQW
       QPWFSERVSD   QUSER       BCH      .0                  SELW
                                                                  More...
Parameters or command
===>
F3=Exit   F5=Refresh       F7=Find      F10=Restart statistics
F11=Display elapsed data   F12=Cancel   F23=More options   F24=More keys
```

Figure 28. WRKACTJOB showing the status of the AS/400 PASE jobs

Chapter 12. Problem determination and messages

From time to time, you will need to debug an application you are creating or gather defect or error information about an application running in your AS/400 PASE environment. This chapter describes the tools that are available, gives small examples of how to use them, and directs you to where in-depth descriptions for each tool may be found.

12.1 Available problem determination tools

There are any number of problem determination tools on both AIX and OS/400. This section only covers the ones that we found useful in producing the samples in this redbook.

12.1.1 Tools on AIX

The tool we found most useful when developing our AIX application was dbx. However, we found that using the printfs function in our C program was as useful as using the dbx debugger. If you choose to use the printf function to assist in debugging your C code in ILE, you should be aware that output is buffered in AIX and in AS/400 PASE. Therefore, you may need to use the `setbuf()` commands to flush your buffers:

```
setbuf(stdout, null);
setbuf(stderr, null);
```

These functions require including stdio.h to function.

Other tools that are available in AIX, which we didn't use, include STEM, core dumps, and any number of ported AIX utilities. More information about the dbx tool can be found on the dbx Tools Web site:
`http://www1.s390.ibm.com:/products/oe/dbx/dbxintoolslinks.v1r1.html`

12.1.2 Tools on OS/400

There are a number of tools we found necessary on OS/400 when looking for messages from our application:

- The job log
- VLOGs (using service tools with maintenance privileges)
- Interactive Debugger
- WRKACTJOB
- WRKSPLF (while compiling our ILE C source)

Any other tool for problem determination that is available on the AS/400 system is also available for use on AS/400 PASE applications:

- SLIC trace
- Operating system traces like trace job
- Performance Explorer traces like TPROF

12.1.2.1 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. Figure 29 shows a screen after a failed compilation of an ILE C module.

```
                          Display All Messages
                                                              System:   AS25B
Job. . :   QPADEV0007    User . . :   SHUPE         Number . . . :   077196

    SHUPE/QCSRC/ILEMOD line 39: Identifier arg2 must be declared before it is
      used.
    SHUPE/QCSRC/ILEMOD line 40: Identifier arg6 must be declared before it is
      used.
    SHUPE/QCSRC/ILEMOD line 40: Identifier arg3 must be declared before it is
      used.
    SHUPE/QCSRC/ILEMOD line 43: Identifier fp must be declared before it is
      used.
    SHUPE/QCSRC/ILEMOD line 62: Identifier icvc must be declared before it is
      used.
    Program ILEMOD is not created because statement errors occurred.
    The compilation failed.
3 > wrksplf
5>> dspjoblog
                                                                        Bottom
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F17=Top   F18=Bottom
```

Figure 29. DSPJOBLOG screen

To view the details of any particular message, you must put the cursor on the message you want to know more about and press the F1 key. Figure 30 shows the details of one of the messages from the first line of the job log shown in Figure 29.

```
                        Additional Message Information

Message ID . . . . . :   CZM0620      Severity . . . . . :   30
Message type . . . . . :   Diagnostic
Date sent  . . . . . :   05/09/00     Time sent  . . . . . :   11:07:25

Message . . . . :   SHUPE/QCSRC/ILEMOD line 39: Identifier arg2 must be
  declared before it is used.
Cause . . . . . :   An error message was generated for line 39 of member
  ILEMOD in file SHUPE/QCSRC.
Recovery  . . . :   Correct all the errors and try the compilation again.  For
  more information see message CZM0041 of message file QCZMSG in library *LIBL
  or the source listing if one has been generated.
Technical description . . . . . :   The compiler attempts to recover
  from the error. Compilation continues.



                                                                       Bottom
Press Enter to continue.

F3=Exit   F6=Print   F9=Display message details   F12=Cancel
F21=Select assistance level
```

Figure 30. Job log message detail

12.1.2.2 VLOGs

VLOGs are issued when a more serious and lower level problem occurs. In general, it is not useful for you to look at VLOGS because they contain specific information about modules, stacks, and variables, which are not easily deciphered. However, if you are going to report a problem to your local support center, you may need to access these logs to save them and send them to IBM.

To access VLOGS, you must be able to start system tools on your AS/400 system. Enter:

```
STRSST
```

Figure 31 on page 176 shows the result of running the STRSST command.

```
                    System Service Tools (SST)

Select one of the following:

     1. Start a service tool
     2. Work with active service tools
     3. Work with disk units
     4. Work with diskette data recovery
     5. Work with system partitions









Selection


F3=Exit         F10=Command entry         F12=Cancel
```

Figure 31. STRSST startup screen

Once the system tools are started, select option **1** then option **5**. At this point, the screen shown in Figure 32 appears.

```
                         Licensed Internal Code Log

Select one of the following:

     1. Select entries from the Licensed Internal Code (LIC) log
     2. Dump entries to printer from the Licensed Internal Code log
     3. Dump entries to media from the Licensed Internal Code log

     5. Change the Licensed Internal Code log sizes
     6. Clear the Licensed Internal Code log
     7. Display the status of the Licensed Internal Code log









Selection


F3=Exit   F12=Cancel
```

Figure 32. Licensed Internal Code Log

From here, you can dump the VLOGs to media or you can display them. If you want to examine them yourself, you can display them but, if you are sending them to IBM, you will want to dump them so an IBM Service Representative can retrieve them from your system. AS/400 PASE generates VLOGS primarily of major code 4700. You can either display any VLOGs of this code type, or you can display all VLOGs for a given time frame. Often, displaying the time frame is more useful since other VLOGs may be generated, which are equally important to problem determination.

If the VLOG was created because of an illegal instruction that was caused by the attempted use of an unsupported syscall, it is possible to scroll right to see the character representation of the VLOG information with the syscall name (the name you would put into the environment variable to work around the unsupported syscall) by pressing PF10. To scroll back to the left, use PF9.

12.1.2.3 WRKACTJOB

WRKACTJOB is a staple tool for examining jobs and job stacks on the AS/400 system. To locate a QP2SHELL job running, you need to scroll down to the QINTER subsystem and find the QP2SHELL job assigned to your user ID or use the `WRKACTJOB SBS(QINTER)` command. See Figure 33.

```
                          Work with Active Jobs                    AS25B
                                                              05/09/00  12:07:27
CPU %:     .0     Elapsed time:   00:00:00     Active jobs:   187

Type options, press Enter.
  2=Change   3=Hold   4=End   5=Work with   6=Release   7=Display message
  8=Work with spooled files   13=Disconnect ...

Opt  Subsystem/Job  User        Type  CPU %  Function        Status
       QPADEV0006   SHUPE       INT      .0  PGM-QP2TERM     DEQW
      +QPADEV0007   SHUPE       INT      .0  CMD-WRKACTJOB   RUN
       QPADEV0009   COOK        INT      .0  CMD-DSPLOG      DSPW
       QP2SHELL     SHUPE       BCI      .0                  TIMW
       QP2SHELL     SHUPE       BCI      .0                  TIMW
       QP2SHELL     SHUPE       BCI      .0                  TIMW
       QP2SHELL     SPARTZ      BCI      .0                  TIMW
     QSERVER        QSYS        SBS      .0                  DEQW
       QPWFSERVSD   QUSER       BCH      .0                  SELW
                                                                    More...
Parameters or command
===>
F3=Exit   F5=Refresh       F7=Find      F10=Restart statistics
F11=Display elapsed data   F12=Cancel   F23=More options   F24=More keys
```

Figure 33. Work with Active Jobs

When you find the particular job you want to examine, you need to enter option 5 next to it to look at job specifics. Further information about WRKACTJOB can be found in *OS/400 Work Management*, SC41-5306, for the release of OS/400 you are using.

12.1.2.4 Interactive Debugger

While developing our samples, which dealt with ILE code, we needed a way to debug an ILE job that was being started from AS/400 PASE. The best tool for this job was the AS/400 Interactive Debugger. The Interactive Debugger is designed to allow you to step through ILE code and examine the contents of variables in your program. A more complete description and example of its use is provided in 12.4, “Debugging the AS/400 to ILE sample” on page 179.

12.1.2.5 WRKSPLF (while compiling our ILE C source)

When you compile ILE programs, you can specify whether you want to print out a listing file. These listings are printed to spooled files, which can be accessed by using the WRKSPLF command. After you execute the command, you arrive at the Work with All Spool Files screen. See Figure 34.

```
                          Work with All Spooled Files

Type options, press Enter.
  1=Send   2=Change   3=Hold   4=Delete   5=Display   6=Release   7=Messages
  8=Attributes        9=Work with printing status


                          Device or                          Total     Cur
Opt  File        User     Queue       User Data    Sts      Pages    Page  Copy
     ILEMOD      SHUPE    PRT01       CRTCMOD      RDY          6            1
     QPSRVDMP    SHUPE    PRT01                    RDY          1            1
     QPSRVDMP    SHUPE    PRT01                    RDY        592            1
     QPJOBLOG    SHUPE    PRT01       QP2FORK      RDY          2            1
     QPJOBLOG    SHUPE    PRT01       QP2FORK      RDY          2            1
     QPJOBLOG    SHUPE    PRT01       QP2FORK      RDY          2            1
     QPJOBLOG    SHUPE    PRT01       QP2FORK      RDY          2            1
     QPJOBLOG    SHUPE    PRT01       QP2FORK      RDY          2            1
     QPJOBLOG    SHUPE    PRT01       QPADEV0009   RDY         15            1
                                                                            Bottom
Parameters for options 1, 2, 3 or command
===>
F3=Exit   F10=View 4   F11=View 2   F12=Cancel   F22=Printers   F24=More keys
```

Figure 34. Work with All Spooled Files screen

From here, you can view a particular spooled file and see the kinds of errors your program might contain.

12.2 Where to find messages on the AIX system

When you are in the development stage of your AS/400 PASE application, you will be working primarily on a system running AIX. For the samples produced in this book, we found the messages that we needed in the terminal session we were compiling in and in the core dump generated by more serious errors. Further messages might be found in other areas on AIX. For further information about AIX messages, refer to the RS/6000 & AIX Support Web page at: `http://duke.toraix.can.ibm.com`

12.3 Where to find messages on the AS/400 system

Once your application is ported to AS/400 PASE, you need to look for error and status messages on the AS/400 system. The most common place to find messages is the job log of the job that is running the QP2TERM or the QP2SHELL. In addition, we found useful messages written to our QP2TERM session. In more extreme cases, you may want to look in the AS/400 system VLOGs for lower level messages.

12.4 Debugging the AS/400 to ILE sample

In this section, we illustrate how we went through debugging our PASE to ILE sample found in 8.3, “Doing callouts to ILE from AS/400 PASE” on page 108. In this example, we needed to debug both AS/400 PASE code and ILE C code running at the same time.

The primary tool for debugging an AS/400 PASE program in AS/400 PASE is the dbx debugger. A detailed description of this tool can be found in the local help on your AIX machine or on the RS/6000 & AIX Support Web site at: `http://duke.toraix.can.ibm.com/lngfiles/dbx.html`

Our sample shows an AS/400 service program being called from an AS/400 PASE program. You need to use the AS/400 Interactive Source Debugger to debug your ILE code. A detailed description of this tool can be found in *Interactive Source Debugger*, SC09-1897.

Once you have both sets of code entered, you must compile your AS/400 PASE program on an AIX system. When you do this, compile errors may be issued either to a specified outfile or to the screen. In either case, these compiler errors are the first steps in debugging your code. Figure 35 on page 180 shows some compiler errors written to an xterm session.

```
.../shupe/PASE> xlc -o PASEtoILE -qldbl128 -qalign=natural
-bI:/afs/rchland.ibm.com/usr1/shupe/PASE/as400_libc.exp PASEtoILE.c

"PASEtoILE.c", line 63.5: 1506-045 (S) Undeclared identifier rc.

.../shupe/PASE>
```

Figure 35. Errors from a xlc compile

Based on the errors, we would then fix our code and FTP it to the AS/400 system on which we're running AS/400 PASE. At this point, we have the binary AIX object, but not necessarily a working ILE program. In our sample, we created an ILE C module, which was then put in a service program. To debug this ILE C module, we need to modify our compile command:

```
CRTCMOD MODULE(MYLIB/MYMODULE)
         SRCFILE(MYLIB/SRCPF)
         OUTPUT(*PRINT)
         DBGVIEW(*ALL)
         TERASPACE(*YES *TSIFC)
```

The added OUTPUT parameter creates a listing file, which can be examined, and `DBGVIEW(*ALL)` creates a debuggable module, which we can access at runtime. You can then create your service program.

We can now debug our sample on the AS/400 system. To debug the entire program, you need two 5250 sessions on your AS/400 system: one to debug the AS/400 PASE program and one to debug your ILE C program.

Once you have the AS/400 PASE terminal session started, start your program in dbx:

```
dbx PASEtoILE
```

By issuing this command in the terminal session, you start a job on your AS/400 system running the AS/400 PASE program QP2FORK. In your second 5250 session, start the `WRKACTJOB` command and scroll down to the QINTER subsystem. In this subsystem, you should now see two QP2FORK jobs assigned to your user ID. Enter option `5` next to the second QP2FORK job (the one with a status of THDW) and take note of the job, user, and number parameters on the resulting screen (Figure 36).

```
                              Work with Job
                                                              System:   AS25B
Job:   QP2FORK        User:   SHUPE          Number:   077376

Select one of the following:

     1. Display job status attributes
     2. Display job definition attributes
     3. Display job run attributes, if active
     4. Work with spooled files

    10. Display job log, if active or on job queue
    11. Display call stack, if active
    12. Work with locks, if active
    13. Display library list, if active
    14. Display open files, if active
    15. Display file overrides, if active
    16. Display commitment control status, if active
                                                                        More...
Selection or command
===>

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel
```

Figure 36. Work with Job screen

You need to start a service job to debug your ILE C program. This is necessary because your ILE C program should have been created with activation group *CALLER and, therefore, will run in the same activation group as the AS/400 PASE program. To debug the program under that job, you need a service job running:

```
STRSRVJOB JOB(077376/SHUPE/QP2FORK)
```

Now, from the AS/400 5250 session on which you started the service job, enter:

```
STRDBG UPDPROD(*YES)
```

Those of you who are familiar with the Interactive Debugger will notice that we did not specify a program to start debugging. This is because service programs cannot have the STRDBG issued against them from the command line. You need to enter command:

```
DSPMODSRC
```

Press F14 or press Shift + F2.

You should now see the Work with Module List screen shown in Figure 37 on page 182.

```
                          Work with Module List
                                                              System:   AS25B
Type options, press enter.
  1=Add program   4=Remove program   5=Display module source
  8=Work with module breakpoints

Opt     Program/module     Library        Type
                           *LIBL          *PGM

  (No programs in source debugger.)









                                                                        Bottom
Command
===>
F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F12=Cancel
```

Figure 37. Work with Module List screen

Enter option 1, the name of the service program, the library it is in, and the type *SRVPGM in the appropriate fields. Press Enter. You should now be able to see the module listed that you want to debug (Figure 38).

```
                          Work with Module List
                                                              System:   AS25B
Type options, press enter.
  1=Add program   4=Remove program   5=Display module source
  8=Work with module breakpoints

Opt     Program/module     Library        Type
         ILEPASE           SHUPE          *SRVPGM
         ILEPASE           SHUPE          *SRVPGM
           ILEMOD2                        *MODULE     Selected









                                                                        Bottom
Command
===>
F3=Exit  F4=Prompt  F5=Refresh  F9=Retrieve  F12=Cancel
Program ILEPASE added to source debugger.
```

Figure 38. Work with Module List: Showing all modules in the service program

Enter option `5` to work with the source code and set your break points. Once the break points are set, return to the first AS/400 5250 session where your terminal session is sitting. You can either choose to run the AS/400 PASE code or step through it. To step through it, type an “s” and press Enter. To run it, type `run` to let the program run. In either case, you eventually come to the _ILECALL statement in your AS/400 PASE program. When you do, the debugger session in your second AS/400 5250 session will activate and display the source line where you set your first breakpoint.

If you are stepping through your AS/400 PASE code, you should come to a screen like the example in Figure 39.

```
                          /QOpenSys/usr/bin/sh

   (dbx)
> s
  stopped in do_init at line 71
     71       init_pid = getpid();
   (dbx)
> s
  stopped in do_init at line 72
     72   }
   (dbx)
> s
  stopped in simple_wrapper at line 160
    160       _ILECALL(ILEtarget,
   (dbx)
> s

===>



 F3=Exit   F6=Print F9=Retrieve F11=Truncate/Wrap F12=Disconnect
F13=Clear F17=Top F18=Bottom F21=CL command entry
 Program ILEPASE added to source debugger.
```

Figure 39. DBX in QP2PASE

At this point, the debug session in your second AS/400 5250 session becomes active and you can step through your code there (Figure 40 on page 184).

```
                          Display Module Source

Program:   ILEPASE        Library:   SHUPE          Module:   ILEMOD2
    39       *  Open a conversion descriptor to convert CCSID 37
    40       *  (EBCDIC) to CCSID 819 (ASCII), that is used for
    41       *  any character data returned to the caller
    42       */
    43      memset(fromcode, 0, sizeof(fromcode));
    44      strcpy(fromcode, "IBMCCSID000370000000");
    45      memset(tocode, 0, sizeof(tocode));
    46      strcpy(tocode, "IBMCCSID00819");
    47      cd = iconv_open(tocode, fromcode);
    48      if (cd.return_value == -1)
    49      {
    50          printf("iconv_open failed\n");
    51          return -1;
    52      }
    53
                                                                        More...
Debug . . .

F3=End program   F6=Add/Clear breakpoint   F10=Step   F11=Display variable
F12=Resume       F17=Watch variable   F18=Work with watch   F24=More keys
Step completed at line 44
Program ILEPASE added to source debugger.
```

Figure 40. Interactive debugger session

You can then evaluate variables in both programs as they are running. When you have finished with your debug sessions, you need to run the ENDDBG and ENDSRVJOB commands in your second AS/400 5250 session. This frees the resources allocated to the debugging effort and reopens the source files for changes.

Chapter 13. Security considerations

From a security point of view, AS/400 PASE programs are subject to the same security restrictions as any other programs on the AS/400 system. To run an AS/400 PASE program on an AS/400 system, 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 accessed by that program, or the program will receive an error when you attempt to access those resources.

13.1 User profiles in AS/400 PASE

Authentication information is stored in individual “profiles” on an AS/400 system rather than in such files as /etc/passwd. Users and groups have profiles. All of these profiles share one name space, so you will not find a user profile named “JOE” and a different group profile also named “JOE.” User and group names are considered to be mono-case. If you pass a lowercase name to the getpwnam() or getgrnam() APIs, the system will fold the case to match the name strings as expected. If you call getpwuid() or getgrgid() to get the profile name returned, it will be in uppercase.

Every user has a uid. Every group has a 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.

The AS/400 system 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, the AS/400 system 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 better security practice to create a specific user ID 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, the AS/400 system does not require group membership for users. The gid of 0 for a user profile on the AS/400 system means “no group assigned” rather than referring to a group with more privileges.

AS/400 security relies on integrated security built into the system from the hardware up. All accesses to objects must pass a security check. The

security check is done with respect to whichever identity the process is running under at the time of the access.

AS/400 PASE relies on address separation for security. If a resource is not available in your address space, you can't access it. File system security prevents someone from loading a resource into their address space. Once in the address space, the resource is available to the process regardless of the identity under which the process is running.

With the advent of IFS, some of this behavior is now also seen in the AS/400 system. For example, when using a stream file, authority is only checked at the time the file is opened and is only checked for the identity under which the process was running at the time of the open. Security checks are not done for additional checks after the open.

AS/400 PASE programs run in a “tags inactive” mode. At the transition to the OS/400 environment, “tags active” is turned on and parameter validation is done. Once in the AS/400 environment, access to traditional AS/400 objects are covered by normal AS/400 security checks.

AS/400 PASE uses the SYSCALL interface when it needs to access resources or behavior normally provided by SLIC. This interface gives AS/400 PASE programs only indirect access to these internals and also appears to be very safe.

13.2 AIX exploitation points

It is relatively difficult to create and spread OS/400-based viruses, because OS/400 programs are protected by program validation checksum. This makes it difficult for someone to distribute patched programs (whether they are user or operating system programs) to other systems without the system administrators of those systems being able to detect those programs as being patched. While not foolproof, it can be used to identify programs not generated by the trusted IBM translator, if the system administrators have their systems properly configured. This is proprietary AS/400 technology.

AS/400 PASE security is more comparable to AIX security, because of the *relative* ease with which AS/400 PASE programs can be patched. AIX-based viruses could function and spread within the AS/400 PASE environment. System administrators should implement all of the standard precautionary measures, including scanning binaries with virus-checking software before transferring them to the AS/400 system.

Chapter 14. AS/400 PASE globalization

Globalization for AS/400 PASE applications is facilitated by the fact that AS/400 PASE runtime is based on AIX runtime. AS/400 PASE programs can use the same rich set of programming interfaces for locales, character string manipulation, date and time services, and character encoding conversions supported on AIX. AS/400 PASE runtime also provides support for application-provided message catalogs (although no message catalogs are shipped in V4R5 for system-supplied runtime and utilities, so any messages from these facilities are presented in English).

AS/400 PASE provides an ASCII programming environment, while most AS/400 system services and data stored on the system use EBCDIC encoding. With only a few exceptions, system support for AS/400 PASE handles ASCII/EBCDIC data conversions automatically for all supported runtime interfaces. The runtime implementation converts character data between the job default (EBCDIC) coded character set identifier (CCSID) and an ASCII (or UTF-8) AS/400 PASE CCSID that is specified as an argument on the ILE Qp2RunPase API. An AS/400 PASE program can also change the AS/400 PASE CCSID using the _SETCCSID runtime function.

14.1 Locale support

AS/400 PASE supports all the interfaces in AIX runtime 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 multi-byte character encodings. AS/400 PASE in V4R5 ships with a subset of AIX 4.3.3 locales, which provide support for a large number of countries and languages using industry-standard encodings (codesets ISO8859-x), codeset IBM-1250, and codeset UTF-8. Euro currency support is provided for applications, which require single-byte encodings with IBM-1252 locales, and for applications that can use multi-byte encodings with UTF-8 locales.

AS/400 PASE does not currently provide locales for PC codesets (such as IBM-850, IBM-943, GBK, and big5) and does not ship a utility to create new locales. But, locales created by customers on an AIX system (using the localedef utility) can be used in AS/400 PASE.

When an AS/400 PASE application changes locales, it generally should also change the AS/400 PASE CCSID (using the _SETCCSID runtime function) to match the encoding for the new locale. This is necessary to ensure that any character data interface arguments are correctly interpreted by AS/400 PASE runtime (and possibly converted when invoking an EBCDIC system service).

It is also necessary for setting the correct CCSID tag on any bytestream file the AS/400 PASE application creates. You can use the cstoccsid runtime function to determine what CCSID corresponds to a codeset name.

AS/400 PASE applications that support Japanese, Korean, Traditional Chinese, and Simplified Chinese are advised to use UTF-8 locales. V4R5 includes other locales for these languages, but the system currently does not support setting the AS/400 PASE CCSID to match the encodings for IBM-eucXX codesets. Using UTF-8 support may require converting file data that may be stored in other encodings (such as Shift-JIS) when the application runs on other platforms.

14.2 File system and sockets support

AS/400 PASE file system and sockets support automatically handles CCSID conversion for character data arguments, except for stream data read from or written to a file descriptor. Character argument data, such as path names and sockets addresses, is converted between the encoding specified by the current AS/400 PASE CCSID and the encodings used internally by AS/400 Integrated File System and sockets services.

The AS/400 Integrated File System tags every bytestream file with a CCSID that is set when the file is created. ILE runtime provides automatic data conversion between the job default CCSID and the bytestream file CCSID for data written to or read from a file opened in text mode. AS/400 PASE runtime does *not* do CCSID conversion for bytestream file data (except for special processing for `stdin`, `stdout`, and `stderr` described in the following section), so the application must do any necessary data conversion. Similarly, AS/400 PASE sockets interfaces do no CCSID conversion for data written to or read from an open socket. An AS/400 PASE application can determine the CCSID of a bytestream file using the STX_XPFSS_PASE command flag on the statx and fstatx runtime functions (an AS/400 extension to AIX runtime support that retrieves an AS/400 stat structure).

Even though AS/400 PASE runtime does no automatic character data conversion of stream data, bytestream files created by an AS/400 PASE application should be correctly tagged so any character data they contain can be correctly handled by ILE runtime. The CCSID tag for any file created by an AS/400 PASE application is set to the current AS/400 CCSID value (specified on the Qp2RunPase API or _SETCCSID function) when the file was created. Applications normally write character data in the same encoding as the locale they are using, so when an AS/400 PASE application changes to a locale with

a different codeset, it should also change the AS/400 PASE CCSID (using the _SETCCSID runtime function) to match the locale encoding.

14.3 stdin, stdout, and stderr

By default, AS/400 PASE runtime connects file descriptors 0, 1, and 2 to the same files used by ILE C for `stdin`, `stdout`, and `stderr`, so any redirection of these ILE C streams also affects `stdin`, `stdout`, and `stderr` for AS/400 PASE programs. The files that ILE C uses for `stdin`, `stdout`, and `stderr` generally contain EBCDIC data. AS/400 PASE runtime provides special support to convert between the AS/400 PASE CCSID and the job default CCSID for data read from file descriptor 0 (`stdin`) and for data written to file descriptors 1 and 2 (`stdout` and `stderr`). This allows an AS/400 PASE application to process ASCII data for `stdin`, `stdout`, and `stderr`, while the files bound to these streams contain EBCDIC data.

The special CCSID conversion support for AS/400 PASE file descriptors 0, 1, and 2 applies only to the file descriptors created when an AS/400 program is invoked by Qp2RunPase. These conversions persist through any descriptor number bound to one of the initial descriptors (for example, when a descriptor is duplicated using the dup runtime function). No conversions are done for any bytestream file, pipe, or socket the application opens (even if they are bound to descriptor numbers 0, 1, or 2).

An AS/400 PASE application can disable automatic CCSID conversion support for file descriptors 0, 1, and 2 by setting ILE environment variables (QIBM_IFS_USE_STANDARD_IO set to Y or I, and QIBM_PASE_DESCRIPTOR_STDIO set to B) before starting the application.

14.4 Interactive terminal support

AS/400 PASE does not currently provide tty device support. However, file descriptors 0, 1, and 2 created when the Qp2RunPase API is invoked are bound to ILE C runtime support for `stdin`, `stdout`, and `stderr`, which provides access to the 5250 display for an interactive job.

In an interactive job, the default ILE C runtime binding for `stdin`, `stdout`, and `stderr` is a simple scrollable list presented on the 5250 display device for the job using the Dynamic Screen Manager. When the Qp2RunPase API or program QP2SHELL is called in an interactive job that uses the default configuration for standard I/O, any data the AS/400 PASE program writes to `stdout` or `stderr` is converted to EBCDIC and written to the 5250 display. And, any data the program reads from `stdin` is the ASCII equivalent of data the

user typed into the input field on the 5250 display. ILE C runtime always does line buffering for `stdin`, `stdout`, and `stderr` in this mode. Therefore, an application may need to write extra newline characters (compared to what is needed for the application running on AIX) to force output to the display, and wait until the user presses the Enter key for any input read from `stdin`.

When you call program QP2TERM to run an AS/400 PASE shell (or any AS/400 PASE program) in a terminal session (managed by the ILE Qp0zRunTerminal API), a similar 5250 display is presented for `stdin`, `stdout`, and `stderr`. AS/400 PASE runtime does CCSID conversion (by default), but the AS/400 PASE program runs in a batch job and uses pipes to communicate with the interactive job. AS/400 PASE programs that run in a terminal session do not need to write extra newline characters to `stdout` or `stderr` to force the output to display, but must wait until the user presses the Enter key for input data read from `stdin`.

14.5 Character encoding conversion support

AS/400 PASE provides automatic character data conversion support for most supported runtime functions. However, applications must handle conversions for stream data written to or read from an open file descriptor and for arguments passed to or from an ILE procedure invoked with the _ILECALL runtime function.

AS/400 PASE runtime does no automatic character argument data conversion when an AS/400 PASE program calls an ILE procedure (using _ILECALL). The application must do any necessary conversions in either the calling program or the called procedure.

ILE runtime and AS/400 PASE runtime provide independent implementations of iconv_open, iconv, and iconv_close to convert between different character encodings. Codeset names used as arguments to the AS/400 PASE version of iconv_open follow AIX naming conventions and are different from the argument values passed to the ILE version of iconv_open. For example, AS/400 PASE codeset ISO8859-1 is the same as ILE encoding IBMCCSID00819.

In addition to support for the iconv set of runtime functions, AS/400 PASE includes the `iconv` utility, which can be invoked from an AS/400 PASE shell to convert bytestream file data.

AS/400 PASE in V4R5 ships a subset of the conversion objects provided by AIX 4.3.3, including conversions for a wide range of EBCDIC, ASCII, UTF-8, and UCS-2 encodings. AS/400 PASE does not currently provide a utility to

create new conversion objects, but conversion objects created by customers on an AIX system (using the `genxlt` utility) can be used in AS/400 PASE.

14.6 Date and time services

AS/400 PASE runtime supports most of the same interfaces as AIX runtime for date and time services, except that AS/400 PASE does not currently provide support to set the system clock. The system clock can be set using standard OS/400 interfaces (such as the SETSYSVAL CL command).

The system time is returned by AS/400 PASE runtime functions, such as gettimeofday is UTC. You must ensure that the system value QUTCOFFSET is set correctly before starting an AS/400 PASE application. AS/400 PASE runtime uses the environment variable TZ to convert UTC to local time, or assume local time is UTC if the TZ variable is not set. OS/400 does not currently store timezone information, so you may want to set ILE environment variable PASE_TZ (or TZ) at the system level so all jobs that call program QP2SHELL to run an AS/400 PASE program use an appropriate default timezone. For example, the following CL command sets the default timezone for all AS/400 PASE programs to Central Time in the USA:

```
ADDENVVAR ENVVAR(PASE_TZ) VALUE(CST6CDT) LEVEL(*SYS)
```

14.7 Database support

AS/400 PASE runtime support for database (through SQL CLI functions) automatically handles ASCII/EBCDIC conversion for all interface elements, with these exceptions:

- Any numeric database field bound as type SQL_CHAR and any database field bound as type SQL_NUMERIC present EBCDIC data to the AS/400 PASE program. For example, a numeric database field with the value 0123 is returned to an AS/400 PASE program through an SQL_NUMERIC binding as 0xf0f1f2f3 (zoned decimal format). AS/400 PASE programs can avoid problems by not using type SQL_NUMERIC and binding numeric fields as types other than SQL_CHAR or SQL_NUMERIC. Alternatively, the application can use iconv services to convert the EBCDIC data to ASCII encoding.
- Pure DBCS (graphic) database fields bound as type SQL_GRAPHIC present EBCDIC DBCS data to the AS/400 PASE program. This restriction will be corrected in a future release. An application can avoid problems by binding pure DBCS database fields as type SQL_WCHAR, which presents the data in UCS-2 encoding.

AS/400 PASE support for database interfaces defaults to using the current (ASCII) AS/400 PASE CCSID when the first database runtime function is invoked by the application. An application can use a different ASCII CCSID by invoking the (AS/400-unique) SQLOverrideCCSID runtime function before invoking any other database interface.

14.8 X-Windows support

AS/400 PASE X-Windows client runtime provides the same support as corresponding AIX X-Windows client runtime (including Motif support). AS/400 PASE does not provide X-Windows server support because the AS/400 system does not support direct attachment of graphical display devices. However, any system that provides X-Windows server support can be used with AS/400 PASE X-Windows application (including a dedicated X-Station, UNIX system, or PC running X-Windows client software).

AS/400 PASE does not provide font server capability, so X-Windows servers attached to an AS/400 PASE application must either have all necessary fonts installed, or must be configured to use another system (generally an RS/6000 system running AIX) as a font server.

14.9 Device support

The AS/400 system does not currently provide support for any hardware devices through files in the /usr directory. AS/400 PASE applications need to call OS/400 interfaces (using the _ILECALL runtime function) to access devices, such as printers and tape drives.

AS/400 PASE support for `stdin`, `stdout`, and `stderr` provide access to the display device in an interactive job. The integrated file system provides a /QOPT file that can be used by AS/400 PASE programs to access files on a CDROM. However, no device-specific controls are possible (such as controlling character display color or ejecting the CD) in either case.

14.10 Shells and utilities

AS/400 PASE in V4R5 includes the Korn, Bourne, and C shells, along with over 100 utilities. The AS/400 system does not provide UNIX-style job control, so AS/400 PASE shells and utilities do not support any functions that rely on job control.

When a shell is invoked from an interactive job by calling program QP2TERM, the shell and any utilities it runs benefit from AS/400 PASE support for automatic ASCII/EBCDIC conversion for `stdin`, `stdout`, and `stderr`. The shell and utilities read and write ASCII data through `stdin`, `stdout`, and `stderr`, even though the 5250 display only handles EBCDIC data. CCSID conversion is only done for data read or written to the 5250 display, so no conversions apply when the output of one utility is piped into another utility. This allows any binary data to be piped between utilities without errors due to character data conversion.

The AS/400 PASE iconv utility can be used to handle character encoding differences that are not automatically handled by the system. For example, the following command line converts an EBCDIC bytestream file to ASCII and then outputs only the last five lines of the file:

```
cat ebcdic.file | iconv -f IBM-037 -t ISO8859-1 | tail -5
```

The AS/400 PASE “system” utility runs any CL command allowed in a batch job. The “system” utility includes support to capture spooled output files and write them to `stdout`, as well as providing ASCII/EBCDIC conversion for any data processed by ILE C through `stdin`, `stdout`, or `stderr` in the CL command processing program. The conversions done by the utility allow the CL command processing program to read and write EBCDIC data through standard I/O, while the shell and any utilities in the pipe see ASCII data. For example, this command line runs the WRKACTJOB CL command and outputs only lines containing the string QPGMR:

```
system wrkactjob | grep QPGMR
```

14.11 AS/400 PASE locales

Table 10 shows the locales shipped with AS/400 PASE in V4R5.

Table 10. Locales shipped with AS/400 PASE

Locale	Language	Territory	CCSID
ar_AA ar_AA.ISO8859-6 AR_AA.UTF-8	Arabic	Arabic Countries	1089 1089 1208
be_BY.ISO8859-5 BE_BY.UTF-8	Byelorussian	Byelorussian SSR	915 1208
bg_BG bg_BG.ISO8859-5 BG_BG.UTF-8	Bulgarian	Bulgaria	915 915 1208

Locale	Language	Territory	CCSID
ca_ES ca_ES.ISO8859-1 ca_ES.IBM-1252 ca_ES.IBM-1252@euro CA_ES.UTF-8 CA_ES.UTF-8@euro	Catalan	Spain	819 819 1252 1252 1208 1208
cs_CZ cs_CZ.ISO8859-2 CS_CZ.UTF-8	Czech	Czech Republic	912 912 1208
da_DK da_DK.ISO8859-1 DA_DK.UTF-8	Danish	Denmark	819 819 1208
de_CH de_CH.ISO8859-1 DE_CH.UTF-8	German	Switzerland	819 819 1208
de_DE de_DE.ISO8859-1 de_DE.IBM-1252 de_DE.IBM-1252@euro DE_DE.UTF-8 DE_DE.UTF-8@euro	German	Germany	819 819 1252 1252 1208 1208
el_GR el_GR.ISO8859-7 EL_GR.UTF-8	Greek	Greece	813 813 1208
EN_AU.UTF-8	English	Australia	1208
EN_BE.UTF-8 EN_BE.UTF-8@euro	English	Belgium	1208 1208
en_GB en_GB.ISO8859-1 en_GB.IBM-1252 en_GB.IBM-1252@euro EN_GB.UTF-8	English	Great Britain	819 819 1252 1252 1208
en_US en_US.ISO8859-1 EN_US.UTF-8	English	United States	819 819 1208
EN_ZA.UTF-8	English	South Africa	1208

Locale	Language	Territory	CCSID
es_ES es_ES.ISO8859-1 es_ES.IBM-1252 es_ES.IBM-1252@euro ES_ES.UTF-8 ES_ES.UTF-8@euro	Spanish	Spain	819 819 1252 1252 1208 1208
ET_EE.UTF-8	Estonian	Estonia	1208
fi_FI fi_FI.ISO8859-1 fi_FI.IBM-1252 fi_FI.IBM-1252@euro FI_FI.UTF-8 FI_FI.UTF-8@euro	Finnish	Finland	819 819 1252 1252 1208 1208
fr_BE fr_BE.ISO8859-1 fr_BE.IBM-1252 fr_BE.IBM-1252@euro FR_BE.UTF-8 FR_BE.UTF-8@euro	French	Belgium	819 819 1252 1252 1208 1208
fr_CA fr_CA.ISO8859-1 FR_CA.UTF-8	French	Canada	819 819 1208
fr_CH fr_CH.ISO8859-1 FR_CH.UTF-8	French	Switzerland	819 819 1208
fr_FR fr_FR.ISO8859-1 fr_FR.IBM-1252 fr_FR.IBM-1252@euro FR_FR.UTF-8 FR_FR.UTF-8@euro	French	France	819 819 1252 1252 1208 1208
hr_HR hr_HR.ISO8859-2 HR_HR.UTF-8	Croatian	Croatia	912 912 1208
hu_HU. hu_HU.ISO8859-2 HU_HU.UTF-8	Hungarian	Hungary	912 912 1208
is_IS is_IS.ISO8859-1 IS_IS.UTF-8	Icelandic	Iceland	819 819 1208

Locale	Language	Territory	CCSID
IT_CH.UTF-8	Italian	Switzerland	1208
it_IT it_IT.ISO8859_1 it_IT.IBM-1252 it_IT.IBM-1252@euro IT_IT.UTF-8 IT_IT.UTF-8@euro	Italian	Italy	819 819 1252 1252 1208 1208
iw_IL iw_IL.ISO8859-8 HE_IL.UTF-8	Hebrew	Israel	916 916 1208
ja_JP ja_JP.IBM-eucJP JA_JP.UTF-8	Japanese	Japan	33722 33722 1208
ko_KR ko_KR.IBM-eucKR KO_KR.UTF-8	Korean	Korea	970 970 1208
LT_LT.UTF-8	Lithuanian	Lithuania	1208
LV_LV.UTF-8	Latvian	Latvia	1208
mk_MK mk_MK.ISO8859-5 MK_MK.UTF-8	Macedonian	Macedonia	915 915 1208
nl_BE nl_BE.ISO8859-1 nl_BE.IBM-1252 nl_BE.IBM-1252@euro NL_BE.UTF-8 NL_BE.UTF-8@euro	Dutch	Belgium	819 819 1252 1252 1208 1208
nl_NL nl_NL.ISO8859-1 nl_NL.IBM-1252 nl_NL.IBM-1252@euro NL_NL.UTF-8 NL_NL.UTF-8@euro	Dutch	Netherlands	819 819 1252 1252 1208 1208
no_NO no_NO.ISO8859-1 NO_NO.UTF-8	Norwegian	Norway	819 819 1208

Locale	Language	Territory	CCSID
pl_PL pl_PL.ISO8859-2 PL_PL.UTF-8	Polish	Poland	912 912 1208
pt_BR pt_BR.ISO8859-1 PT_BR.UTF-8	Portuguese	Brazil	819 819 1208
pt_PT pt_PT.ISO8859-1 pt_PT.IBM-1252 pt_PT.IBM-1252@euro PT_PT.UTF-8 PT_PT.UTF-8@euro	Portuguese	Portugal	819 819 1252 1252 1208 1208
ro_RO ro_RO.ISO8859-2 RO_RO.UTF-8	Romanian	Romania	912 912 1208
ru_RU ru_RU.ISO8859-5 RU_RU.UTF-8	Russian	Russia	915 915 1208
sh_SP sh_SP.ISO8859-2 SH_SP.UTF-8	Serbian Latin	Yugoslavia	912 912 1208
sk_SK sk_SK.ISO8859-2 SK_SK.UTF-8	Slovak	Slovakia	912 912 1208
sl_SI sl_SI.ISO8859-2 SL_SI.UTF-8	Slovene	Slovenia	912 912 1208
sq_AL sq_AL.ISO8859-1 SQ_AL.UTF-8	Serbian Cyrillic	Yugoslavia	915 915 1208
sr_SP.ISO8859-5 SR_SP.UTF-8	Serbian Latin	Yugoslavia	915 1208
sv_SE sv_SE.ISO8859-1 SV_SE.UTF-8	Swedish	Sweden	819 819 1208
TH_TH.UTF-8	Thai	Thailand	1208

Locale	Language	Territory	CCSID
tr_TR tr_TR.ISO8859-9 TR_TR.UTF-8	Turkish	Turkey	920 920 1208
UK_UA.UTF-8	Ukrainian	Ukraine	1208
VI_VN.UTF-8	Vietnamese	Vietnam	1208
zh_CN zh_CN.IBM-eucCN ZH_CN.UTF-8	Chinese (simpl)	China	1383 1383 1208
zh_TW zh_TW.IBM-eucTW ZH_TW.UTF-8	Chinese (trad)	Republic of China	964 964 1208

14.12 AS/400 PASE codesets

Table 11 shows the codeset values supported for iconv conversions in AS/400 PASE V4R5. Any characters in the source encoding that are not supported in the target encoding are converted to substitution characters.

Table 11. Supported iconv conversions

Codeset name	Description
UTF-8	Universal Transfer Format
IBM-037	COM EUROPE EBCDIC
IBM-1148	INTL ECECP
IBM-500	INTL EBCDIC
IBM-1140	COM EUROPE ECECP
IBM-850	LATIN-1 PC-DATA
IBM-852	LATIN-2 PC-DATA
IBM-1252	MS-WIN LATIN-1
IBM-1145	SPANISH ECECP
IBM-284	SPANISH EBCDIC
IBM-1146	UK ECECP
IBM-285	UK EBCDIC

Codeset name	Description
IBM-860	PORTUGESE PC-DATA
IBM-1147	FRENCH ECECP
IBM-297	FRENCH EBCDIC
IBM-861	ICELAND PC-DATA
IBM-437	USA PC-DATA
IBM-863	CANADA PC-DATA
IBM-865	DEN/NORWAY PC-DAT
IBM-1141	AUS/GERM ECECP
IBM-273	AUS/GERM EBCDIC
IBM-1142	DEN/NORWAY ECECP
IBM-277	DEN/NORWAY EBCDIC
IBM-1143	FIN/SWEDEN ECECP
IBM-278	FIN/SWEDEN EBCDIC
IBM-1144	ITALIAN ECECP
IBM-280	ITALIAN EBCDIC
IBM-1122	ESTONIA EBCDIC
IBM-922	ESTONIA ISO-8
IBM-921	Baltic - Multilingual
IBM-1112	BALTIC EBCDIC
IBM-869	GREEK PC-DATA
IBM-875	GREEK EBCDIC
IBM-857	TURKISH PC-DATA
IBM-1026	TURKEY LATIN-5 EB
IBM-1046	ARABIC - PC
IBM-856	HEBREW PC-DATA
IBM-932	JAPAN MIX PC-DATA
IBM-942	JAPAN MIX PC-DATA

Codeset name	Description
IBM-930	JAPAN MIX EBCDIC
IBM-939	JAPAN MIX EBCDIC
IBM-934	KOREA MIX PC-DATA
IBM-944	KOREA MIX PC-DATA
IBM-933	KOREA MIX EBCDIC
IBM-946	S-CHINESE PC-DATA
IBM-936	S-CHINESE PC-DATA
IBM-1381	S-CH GB PC-DATA
IBM-935	S-CHINESE MIX EBC
IBM-836	S-CHINESE EBCDIC
IBM-837	S-CHINESE EBCDIC
IBM-948	T-CHINESE PC-DATA
IBM-938	T-CHINESE MIX PC
IBM-937	T-CHINESE MIX EBC
IBM-eucKR	EUC Korean
IBM-eucJP	EUC Japanese
IBM-eucCN	EUC Simplified Chinese
IBM-eucTW	EUC Traditional Chinese
ISO8859-1	ISO Latin-1
ISO8859-2	ISO Latin-2
ISO8859-3	ISO Latin-3
ISO8859-4	ISO Latin-4
ISO8859-5	ISO Latin-5
ISO8859-6	ISO Latin-6
ISO8859-7	ISO Latin-7
ISO8859-8	ISO Latin-8
ISO8859-9	ISO Latin-9

Codeset name	Description
Note: ECECP means Euro ready.	

Appendix A. Programming resources

Table 12 lists the files that are available for application development and porting to AS/400 PASE.

Table 12. Programming resource locations

File	Reason	Where
as400_protos.h	AS/400 PASE to ILE	`http://www.ibm.com/as400/developer/factory/pase/index.html` or AS/400 Software Support
as400_types.h	Unique AS/400 parameter types for calls to ILE	See above
as400_libc.exp	Link export file for AS/400 unique functions in libc.a	See above
libdb400.exp	Link export file for AS/400 database access	See above
qp2user.h	ILE to AS/400 PASE	See above
qp2shell.h	ILE to AS/400 PASE	See above
qp2term.h	ILE to AS/400 PASE	See above
sqlcli.h	Access to AS/400 database header file	Available in QSYSINC/H on the AS/400 system, and /QIBM/include/sqlcli.h within IFS (the OS/400 System Openness Includes (Product Option 13) needs to be installed on the AS/400 system to get this include file)

Appendix B. The Application Factory

Could your AS/400-based applications be enhanced and complemented by key applications that currently reside on Windows NT or UNIX systems? Are you interested in porting additional functions that you presently deliver on other platforms to your AS/400 solution? Are you bringing new Java functions to your application? If you answered “yes” to any of these questions, the Application Factory project recently initiated by PartnerWorld for Developers, AS/400, is designed to help you. An overview of the site is provided in this appendix. Be sure to visit the site online at:
`http://www.ibm.com/as400/developer/factory/`

B.1 Overview

The Application Factory is designed to help IBM Business Partners provide attractive solutions more quickly to the AS/400 market. In short, the Factory is an organization of information and tools to help the PartnerWorld team work most effectively with you in bringing solutions to the AS/400 platform. The Application Factory can help you estimate delivery of a solutions port. It can also assist you in developing a checklist to factor in AS/400 customer expectations regarding your new development efforts. The Application Factory uses the Web to deliver evolving porting information and experiences. The fundamental goal of the Application Factory is to:

- Organize existing application and porting information so you can find it faster and use it more effectively.
- Identify areas where PartnerWorld for Developers, AS/400, should create new documentation, tools, wizards, or support structures to facilitate the delivery of your solutions on the AS/400 platform.

The organization of the Application Factory comes from the solution characteristics that have long been important to AS/400 customers and from IBM’s commitment to support you in providing solutions with these characteristics. AS/400 customers in general are very loyal to the platform and to those solutions that deliver these AS/400 values. We want to make it easy for you to continue to build these values into your future solutions.

The Application Factory divides the work of creating an AS/400 solution into several parts. Separating the discussion into these parts can help you more accurately estimate the short term coding and longer term support costs for a particular solution. The Factory provides a strong vehicle for PartnerWorld for Developers, AS/400, to provide you with better tools in key areas. For ported solutions, the Factory also helps distinguish porting costs and technologies

from the activities of specifically “creating an AS/400 product” — regardless of the porting technology used (for example ILE, Integrated Netfinity Server, or AIX integrated runtime AS/400 PASE).

So, how does the Application Factory work logistically? Well first, let’s state the obvious: “One size does not fit all.” The structure of solutions and the needs of partners and their customers vary. So, the Factory is intended to provide a checklist from which you can see that the relevant porting items are covered, and that all the right questions are asked in bringing a solution to the AS/400 market.

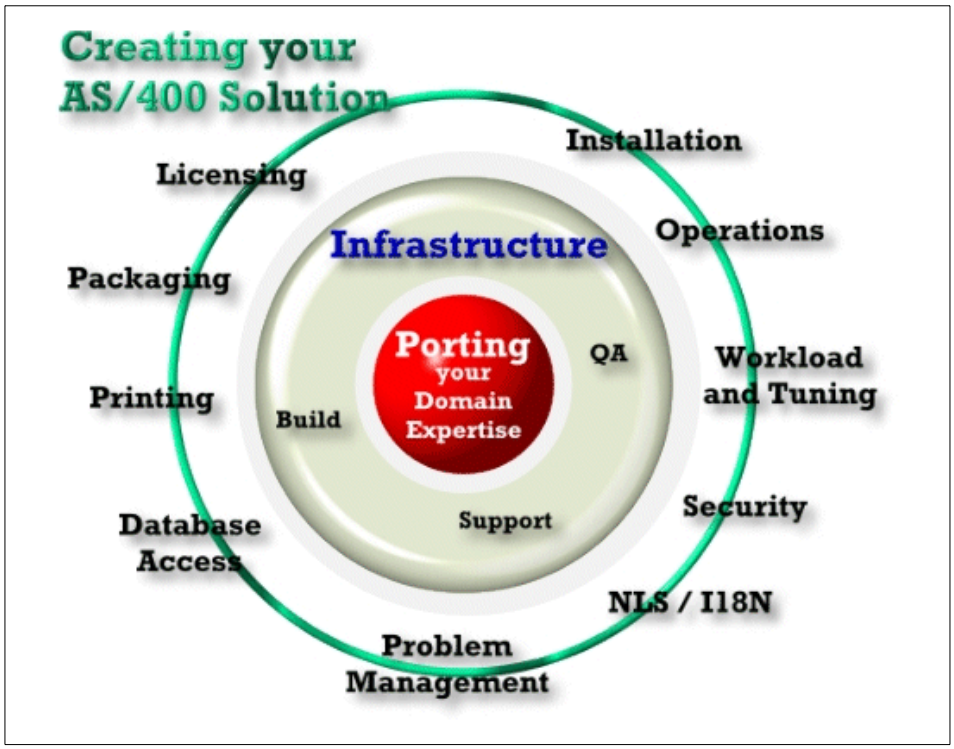


Figure 41. Creating your AS/400 solution

The first Factory “area of expertise” involves core solution functionality. This is where your domain expertise and business value is delivered to customers. The types of support we focus on here include product creation tools, such as those in the AS/400 AD Tools Network, and the various porting technologies.

The next area of expertise relates more to process or infrastructure. This area encompasses items, such as acquiring hardware and software, establishing

product build procedures, incorporating quality assurance and testing, and training your AS/400 product support staff. These items are usually the ongoing investments in supporting a platform.

The third Factory area of expertise regards those tasks that may be required to specifically adapt an application for the AS/400 world — efforts that put familiar faces on core solution functions for current AS/400 customers. Generally, these components are done once and can then be reused for future releases, so they are typically cost-contained.

B.2 Around the circle

In Figure 41, you see an outer circle of items for porting. In this section, we outline each of those items.

B.2.1 Licensing

AS/400 provides facilities that permit you to license your software product in the same manner in which IBM AS/400 products are licensed. These interfaces are provided as system APIs. For more information, see 2.3, “Capabilities” on page 14.

B.2.2 Installation and packaging

There are several means by which customers may install their AS/400 solution. They range from UNIX-like scripting to more fully integrated installation and packaging similar to AS/400 system software.

You may choose to distribute your solution on CD-ROM. You need to acquire the necessary hardware and software to premaster and record a CD-ROM from your AS/400 data. More information about using CD-ROMs for distributing AS/400 software can be found on the Web at:
`http://www.ibm.com/as400/optical/cdrom/cddist.html`

Several solutions are available for purchase:

- Vendor: Agent Data ApS
  Product: BlueCD
  Distributed in the U.S. by Kisco Information Systems
- Vendor: Bug Busters Software Engineering
  Product: CD Builder

- Vendor: Centerfield Technology, Inc. (formerly Bradenmark Software)
  Product: CD-ROM Studio for the AS/400
- Vendor: Mid-Comp International
  Product: CD Mastering Software

B.2.3 Operations

There are two types of operator interfaces on the AS/400 system today: “green screen” menus and commands and graphical. Both can now be extended to provide application operations interfaces. In addition, several scripting facilities are available on the system to facilitate system operations.

The menus and command lines are the long-standing AS/400 interfaces and are still used by many customers. While not flashy, they facilitate ISVs and customers writing operational scripts that can automate application use and minimize operator interventions, lowering the cost of operations for a particular application. The AS/400 Command Language (CL), REXX, or the QShell Interpreter (an EBCDIC shell), can be used for scripting the commands.

The graphical interface was released for the first time several years ago and is steadily growing in popularity. With new plug-in capabilities, applications can add their operational interfaces to this system view also. The Operations Navigator infrastructure now supports Java and Visual Basic Plug-ins, as well as C++ plug-ins. In V4R4, a new toolset was added to make developing graphical user interfaces for Java easier (GUIBuilder, Resource Script Converter, and PCML to get or set data to AS/400 systems).

Scripting resources include:

- OS/400 CL programming
- Creating AS/400 commands:
  `http://www.ibm.com/as400/developer/porting/400cmds.html`
- REXX programming
- Shell scripting via Qshell Interpreter:
  `http://www.ibm.com/as400/developer/qshell`

Graphical interface resources include:

- AS/400 Operations Navigator Plug-in Support Web page:
  `http://www.ibm.com/as400/oper_nav/pluginpage.htm`
- Tech Studio references:
  `http://www.ibm.com/as400/tstudio/opsnav/plugin/pludex.htm`

Or, consider the IBM Redbook *AS/400 Client Access Express for Windows: Implementing V4R4M0*, SG24-5191.

B.2.4 AS/400 workloads and performance

OS/400 provides many services for scheduling work and monitoring how the system is running. An understanding of this information will help you leverage these built-in capabilities to fine tune your solutions and increase your customers' satisfaction.

B.2.5 Security

AS/400 system security is among the best in the industry. It is rooted in the AS/400 object-based architecture itself. This architecture provides integrated traditional OS/400 and UNIX system security functions and facilities. Some specific considerations for security ports are included in the security section of the porting FAQs.

To understand these facilities and the AS/400 security perspective, see Chapter 13, “Security considerations” on page 185, or visit the OS/400 security-related FAQs Web site at:
`http://www.ibm.com/as400/developer/porting/faq.html#header_11`

B.2.6 National language and internationalization

Support for worldwide customers to work in their native languages has been a priority for the AS/400 system since its inception. Industry terms relating to this functional area, including internationalization (I18N), national language support (NLS), national language versions (NLV), localization, and globalization are supported on different operating systems in different ways.

AS/400 supports multiple models—a powerful, long-standing model that tags all data on the system with its encoding, plus message catalog support standardized for UNIX systems. The following references are useful in understanding this support:

- *PWD/400 DBCS Guide*:
  `http://www.ibm.com/as400/developer/dbcs/guide.html`
- *AS/400 International Application Development*, SC41-5603
- *National Language Support*, SC41-5101

In addition, because the AS/400 system is historically based on the EBCDIC character set, there are some other potential considerations for working with application data that may already reside on the system.

B.2.7 Problem management

Problem management, in AS/400 terms, includes learning how to retrieve and respond to messages. Messages may be found in one or more queues or logs:

- QSYSOPR (system operator) message queue
- Worstation (user) message queue
- Job log
- History log

A good place to start learning about the details of AS/400 problem management is the manual *AS/400 Basic System Operation, Administration, and Problem Handling*, SC41-5206.

To further customize your solution to capture and report problems in an AS/400 manner, consider the OS/400 Problem Management APIs.

B.2.8 Database access

DB2 Universal Database for AS/400 is delivered with every AS/400 system. It is fully integrated with the OS/400 for security, save/restore operations, and other system support functions.

For general information about DB2 UDB for AS/400, see:

- DB2 Universal Database for AS/400:
  `http://www.ibm.com/as400/db2/db2main.htm`
- Database Porting FAQs:
  `http://www.ibm.com/as400/developer/porting/faq.html#Header_7`

B.2.9 Printing

The AS/400 system provides a print spooling subsystem that can manage printing for you. To learn how to leverage AS/400-specific print facilities, you may want to review the following manuals:

- *OS/400 Printer Device Programming V4R4*, SC41-5713
- *PrintManager API Reference*, S544-3304

Appendix C. AS/400 PASE compared to ILE

Check the table at the following Web site for the most recent assessment of the relative differences between PASE and ILE:
`http://www.ibm.com/as400/developer/factory/table.html`

A snapshot of the tables is included below, using the keys described in Table 13.

Table 13. Key

Symbol	Meaning
++	Very favorable; little to no effort required
+	Favorable; minor effort required
o	Neutral
-	Significant effort required
--	Very significant effort required
+/-	Effort varies based upon analysis

Table 14 outlines the general AS/400 issues that arise when porting or migrating an application to OS/400.

Table 14. General considerations

Porting considerations	ILE		AS/400 PASE	
AS/400 hardware support	+ +	All current hardware supported	+	All e-Series hardware supported (post 9/97)
OS/400 software support	+ +	All current releases supported	o	V4R4 and later
Use of UNIX system interfaces	+	Some UNIX or POSIX APIs are supported	+ +	Many UNIX and POSIX APIs supported and AIX extensions supported
Process for assessing portability	+	API analysis first, then ASCII dependencies and pointer changes	+ +	API Analysis is a reasonable estimator of portability
Generate machine code	- -	Architecture does not support	+ +	Supported
Use specific directly attached devices	-	Workaround dependency	-	Workaround dependency

Porting considerations	ILE		AS/400 PASE	
Compute intensive applications	o	Math libraries not optimized	+ +	Optimized math libraries
Staff has AS/400 skills	+ +	Skills may already be ILE, or adding ILE skills will be a benefit to programmers	+	Minimal AS/400 skills needed for one time “productization” of product
Complicated UNIX build process	o	Use NFS to deliver source for compilation in ILE, map a network drive, or use REXEC	+ +	Can use existing AIX build structures
Extensive test harness for quality assurance	+	Use NFS to deliver test results for analysis, or map network drive	+ +	Use NFS, shells, and utilities

Table 15 outlines issues that arise due to the platform on which the application resides before being ported.

Table 15. Currently supported platforms

Porting considerations	ILE		AS/400 PASE	
Release to release compatibility required	+ +	If “observability” (AS/400 intermediate compilation form) is maintained, hardware change will not require recompilation	+	Similar to AIX upgrade; recompilation required for new hardware architecture, not necessarily for processor upgrades
Existing AIX version of application	+ / -	Analysis for porting process will determine difficulty; database use will require changes	+ +	Database use will require changes; other changes minimized
Application runs on AIX 4.3.3	+ / -	Analysis for porting process will determine difficulty	+ +	AS/400 PASE targets this AIX release; for earlier releases, may require recompilation on appropriate AIX release
Application is non-AIX UNIX	o	Need AS/400 skills for both port and creating AS/400 product; education for new customer base	+	Need AIX skills for port; need AS/400 skills for “productization”; education to support new customer base

Table 16 outlines the application structure and integration issues that must be considered.

Table 16. Application structure and integration

Porting considerations	ILE		AS/400 PASE	
Integration w/ILE code outside application	+ + +	No change required	+	Calls to ILE supported but should be limited; possible performance issues for large ASCII to EBCIDIC conversions
Application is extendable by customer	+ + +	AS/400 development tools are available to build callable parts, some as/400 hosted, some cross-platform tools	+	Require AIX system for customer builds or integration of ILE calls into application to connect to customer code
Application is organized in a UNIX library structure	o	If function is to be called in ILE application, adapt library to AS/400 ILE service programs	o	If function is to be called by PASE application code, port shared libraries to PASE
Application function uses a server/daemon process model	+ + +	Supported	+ + +	Supported

Table 17 provides a comparison between ILE and AS/400 PASE ports when specific system services are used.

Table 17. Application interaction with system services

Porting considerations	ILE		AS/400 PASE	
Dependency on fork or exec (not in combination)	-	Change to use spawn and adjust entry point	+ + +	Supported
Dependency on setuid/setgid independent of user and group credentials	o	Change to use swap function; or AS/400 version of setuid/setgid functions	+ + +	Supported
Dependency on X Windows support	+	Use third-party products for EBCDIC support	+ + +	Supported

Porting considerations	ILE		AS/400 PASE	
A single environment for both development and deployment of applications required	+	Available both with AS/400 or third- party tools for ILE	o	AIX development environment; AS/400 deployment
Dependency on ASCII codeset	+	ASCII C runtime PRPQ is available; other interfaces require EBCDIC	+ +	Supported
64 bit support (4/8/8 model)	- -	For files not for C runtime For processes, recode to use teraspace	+ + -	For files supported For processes, currently only 32-bit address space supported

Table 18 provides a comparison between ILE and AS/400 PASE when specific languages are being used.

Table 18. Language considerations

Porting considerations	ILE		AS/400 PASE	
Application coded in C	+	ANSI standard C with access to both stream and database files	+ +	AIX compilation required using AIX compilers Non-ANSI practices are supported
Application coded in C++	+	ANSI C++ draft standard X3J16/95-001 supported	+ +	Supports full ANSI C++ VisualAge C++
Application coded in Fortran	- -	unsupported	+ +	AIX Fortran runtime
Application coded in COBOL	+	ANSI 85 standard COBOL supported with AS/400 extensions	- -	Unsupported
Application relies on shell programming	+	QSHELL interpreter	+ +	QSHELL, QP2SHELL, and QP2TERM; PASE, Kern, Bourne, and Cshell

Porting considerations	ILE		AS/400 PASE	
Development relies on shell programming	+ 	QSHELL interpreter	+ +	QSHELL, QP2SHELL and QP2TERM supported; PASE, Kern, Bourne, and Cshell

Table 19 provides a comparison between ILE and AS/400 PASE when database porting is being considered.

Table 19. Database considerations

Porting considerations	ILE		AS/400 PASE	
Database access using typical UNIX database	o	DB2 UDB for AS/400 supported along with native AS/400 interfaces; port required	o	DB2 UDB for AS/400 supported; port required
Application uses embedded SQL	+	Port to DB2 UDB for AS/400 and use precompiler	-	Embedded SQL is not available. Extract major db interaction logic into stored procedures and call from PASE
Application uses CLI	+	Available, some EBCDIC assumptions; port to DB2 UDB for AS/400	+	Available, returns ASCII data; port to DB2 UDB for AS/400
Application performs many DB transactions	+ +	Many fine grained requests supported through proprietary interfaces	+	Preferred fewer, larger requests if ASCII translation required

Table 20 provides a comparison between ILE and AS/400 PASE when third-party products are part of the porting consideration.

Table 20. Third-party product interaction

Porting considerations	ILE		AS/400 PASE	
Application has middleware requirements	o	If most requirements met by ILE, port to ILE then access PASE for additional middleware	o	If most requirements met by PASE, port to PASE then access ILE for additional middleware

Porting considerations	ILE		AS/400 PASE	
Dependency on larger partner application as primary data source	o	Consider where the major application runs and latency between application port under consideration and major application	o	Consider where the major application runs and latency between application port under consideration and major application

Appendix D. Special notices

This publication is intended to help IBM Business Partners, Independent Software Vendors and IBM Global Service Professionals to prepare, plan and perform UNIX application porting to the AS/400 system. The information in this publication is not intended as the specification of any programming interfaces that are provided by OS/400 V4R5 and option 33. See the PUBLICATIONS section of the IBM Programming Announcement for OS/400 V4R5, 5796-SS1, for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

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

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate

them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

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

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

AIX	AS/400
AS/400e	AT
C/400	CT
Current	DB2
DB2 Universal Database	Hummingbird
IBM®	Integrated Language Environment
Language Environment	Manage. Anything. Anywhere.
Netfinity	OS/400
PAL	PartnerWorld
PrintManager	RS/6000
SP	System/390
VisualAge	Visualization Data Explorer
XT	400
Lotus	Domino

Tivoli	TME
NetView	Cross-Site
Tivoli Ready	Tivoli Certified

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Appendix E. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 225.

- *AS/400 Client Access Express for Windows: Implementing V4R4M0*, SG24-5191
- The following publications are available online in softcopy format only at the IBM Redbooks homepage at: `http://www.redbooks.ibm.com/`

  At the site, click **Redbooks Online** and then enter the publication number or title in the search field that appears. Click **Submit Search** and select the appropriate publication title that appears.

  - *UNIX C Applications Porting to AS/400*, SG24-4438
  - *UNIX C Applications Porting to AS/400 Companion Guide*, SG24-4938

E.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at `http://www.redbooks.ibm.com/` for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

E.3 Other resources

These publications are also relevant as further information sources:

- *PrintManager API Reference*, S544-3304
- *Interactive Source Debugger*, SC09-1897
- *National Language Support*, SC41-5101
- *AS/400 Basic System Operation, Administration, and Problem Handling*, SC41-5206
- *OS/400 Work Management*, SC41-5306
- *AS/400 International Application Development*, SC41-5603
- *ILE C/400 Programmer's Reference,* SC41-5607
- *Integrated File System Introduction*, SC41-5711
- *OS/400 Printer Device Programming V4R4*, SC41-5713
- *OS/400 Network File System Support*, SC41-5714
- *DB2 UDB for AS/400 SQL Call Level Interface (ODBC)*, SC41-5806
- *OS/400 UNIX-Type APIs V4R4*, SC41-5875

  This publication is available in softcopy format only from the AS/400 Online Library at:
  `http://as400bks.rochester.ibm.com/pubs/html/as400/onlinelib.htm`

  At the site, select your language and click **GO!** Select **V4R4** and then click **Search or view all V4R4 books**. In the search field that appears, enter the publication number or title and click **Find**. Select the appropriate publication title that appears.

E.4 Referenced Web sites

These Web sites are also relevant as further information sources:

- The GNU Open Source Web site can be found at: `http://www.gnu.org/`
- The GNU Open Source FTP site can be found at: `ftp://ftp.gnu.org/`
- The OpenDX Web site can be accessed at: `http://www.opendx.org/`
- Visit the AS/400 Custom Technology Center online at:
  `http://www.ibm.com/as400/Service/welcome_3.htm`
- Access the Hummingbird Exceed Windows-based X server at:
  `http://www.hummingbird.com/`

- Access the X-Win32 Windows-based X server at:
  http://www.starnet.com/productinfo/
- The AS/400 QShell Interpreter can be found online at:
  http://www.ibm.com/as400/developer/qshell
- The popular midrange mailing list, Midrange-L, which is implemented using Perl, can be accessed online at: http://www.midrange.com
- An API analysis tool can be accessed online at:
  http://www.ibm.com/as400/developer/porting/apitool.html
- Application providers can obtain support for porting their applications to AS/400 PASE from PartnerWorld for Developers at:
  http://www.ibm.com/as400/developer/factory/
- GNOME, part of the GNU project, is free (open source) software that can be accessed at: http://www.gnome.org
- Organizational, legal, and financial support for the Apache open-source software projects can be viewed online at: http://www.apache.org
- The GNU Image Manipulation Program (GIMP) is freely distributed software suitable for such tasks as photo retouching, image composition, and image authoring. It is available online at: http://www.gimp.org
- GTK+, an Open Source Free Software GUI Toolkit, primarily developed for use with the X Window system, is available online at: http://www.gtk.org
- IceWM, a window manager for the X11 Window System, is available online at: http://icewm.sourceforge.net
- Perl, a high level programming and premier Web scripting language, can be downloaded as an open source at: http://www.perl.com
- Virtual Network Computing (VNC) is a remote display system which allows you to view a computing 'desktop' environment from anywhere on the Internet and from a wide variety of machine architectures. It is available online at: http://www.uk.research.att.com/vnc
- The DBI Perl module can be accessed online at:
  http://www.perl.org/CPAN/modules
- Chimera, a World Wide Web browser for UNIX-based machines running the X window system, is available online at:
  http://www.cs.unlv.edu/chimera
- GNU Wget, a freely available network utility for retrieving files from the World Wide Web using HTTP and FTP, can be accessed at:
  http://www.gnu.org/software/wget/wget.html

- A list of current PTFs can be found at:
  http://www.ibm.com/as400/developer/factory/pase/misc.html
- Acces the Qshell Utility Lists online at:
  http://www.as400.ibm.com/developer/qshell/utils.html
- Considerations for ILE and AS/400 PASE porting are located at:
  http://www.as400.ibm.com/developer/factory/table.html
- A detailed description of the dbx debugging tool can be found on the RS/6000 & AIX Support Web site at:
  http://duke.toraix.can.ibm.com/lngfiles/dbx.html
- Information about the dbx debugging tool can be found on the dbx Tools Web site at:
  http://www1.s390.ibm.com:/products/oe/dbx/dbxintoolslinks.v1r1.html
- For OS/400 security-related frequently asked questions, log on to:
  http://www.ibm.com/as400/developer/porting/faq.html#header_11
- Access the *PWD/400 DBCS Guide* online at:
  http://www.ibm.com/as400/developer/dbcs/guide.html
- For information about creating AS/400 commands, visit the Web site at:
  http://www.ibm.com/as400/developer/porting/400cmds.html
- Visit the AS/400 Operations Navigator Plug-in Support Web page at:
  http://www.ibm.com/as400/oper_nav/pluginpage.htm
- Access Tech Studio references online at:
  http://www.ibm.com/as400/tstudio/opsnav/plugin/pludex.htm
- For general information about DB2 Universal Database for AS/400, log on to: http://www.ibm.com/as400/db2/db2main.htm
- For frequently asked questions regarding Database Porting, visit the Web site at: http://www.ibm.com/as400/developer/porting/faq.html#Header_7

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** http://www.redbooks.ibm.com/

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States	usib6fpl@ibmmail.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name	Last name	
Company		
Address		
City	Postal code	Country
Telephone number	Telefax number	VAT number

☐ Invoice to customer number _____

☐ Credit card number _____

Credit card expiration date	Card issued to	Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

ABI See *Application Binary Interface.*

Advanced Interactive eXecutive (AIX) An integrated UNIX operating environment. Most of the references in this book are for the RS/6000 version of AIX.

AFS See *Andrew File System.*

AIX See *Advanced Interactive eXecutive.*

American National Standard Code for Information Interchange (ASCII) A code page usually used on PCs and UNIX systems.

Andrew File System (AFS) Distributed file system developed by IBM and CMU, now marketed by Transarc Corp.

API Application Programming Interface.

Application Binary Interface (ABI) The allowable machine instructions, register conventions, stack layout, and argument/parameter linkage required to interoperate with other programs and a system that uses the ABI.

AS/400 Portable Application Solutions Environment (AS/400 PASE) An AS/400 runtime environment that delivers a broad subset of the AIX interface on the AS/400 system.

ASCII See *American National Standard Code for Information Interchange*.

CRM See *Customer Relationship Management.*

Customer Relationship Management (CRM) Generally used for the business applications that are used to manage customer relationships.

daemon A task, process, or thread that intermittently awakens to perform some tasks and then goes back to sleep (software). Frequently used to describe management or server tasks in UNIX.

Data Description Specifications (DDS) Used in OS/400 to describe data, including databases, screens, and printer files.

DDS See *Data Description Specifications.*

DFS See *Distributed File System.*

Distributed File System (DFS) An OSF DCE component.

EBCDIC See *Extended Binary Coded Decimal Interchange Code.*

Enterprise Resource Planning (ERP) *G*enerally used for core business applications.

ERP See *Enterprise Resource Planning.*

Extended Binary Coded Decimal Interchange Code (EBCDIC) A code page usually used on AS/400 systems.

File Transfer Protocol (FTP) A standard protocol used to transfer data files between systems over TCP/IP.

FTP See *File Transfer Protocol*.

GNU GNU's Not UNIX. The Free Software Foundation's UNIX-like operating system project.

GPL GNU General Public License.

IFS See *integrated file system*.

ILE See *integrated language environment*.

IMPI See *Internal Microprogramming Interface*.

ITSO International Technical Support Organization.

integrated file system (IFS) The hierarchical directory and file system for AS/400 systems. AS/400 “libraries” and the objects they contain can be accessed as a mounted file system (QSYS.LIB) under the IFS root.

integrated language environment (ILE) The AS/400 program execution environment that

supports bound procedures. All C application programs on the AS/400 system run in ILE.

Internal Microprogramming Interface The original processor architecture of the AS/400 system. IBM no longer manufactures IMPI-based AS/400 systems.

LGPL Library GNU public license.

MI See *Machine Interface (architecture)*.

Machine Interface (MI) (architecture). MI architecture is the high-level abstract machine definition implemented by AS/400 systems. Also referred to as Technology-Independent Machine Interface (TIMI).

Network File System (NFS) A standard protocol for mapping file drives over TCP/IP, developed by USA Sun Microsystems Inc.

NFS See *Network File System.*

NGPL Nethack GNU public license.

NLS National language support.

PASE See *AS/400 Portable Application Solutions Environment* (AS/400 PASE).

PLS See *Process Local Storage*.

Process Local Storage (PLS) The part of an address space that is private to a process. An application program can control what memory is mapped to a particular address in PLS, and can specify different storage protection for multiple mappings of the same memory. All PASE memory is equivalent to PLS.

OLTP See *on-line transaction processing.*

on-line transaction processing (OLTP). Used to refer to transactional business applications.

OSF DCE Open Systems Foundation Distributed Computing Environment.

POSIX See *Portable Operating System Interface.*

Portable Operating System Interface (POSIX). For computer environments, an IEEE operating system standard, closely related to the UNIX system (software writing).

productization The process by which an application is made into an AS/400 installable product.

SCM See *Supply Chain Management.*

Simple Message Block (SMB) A standard protocol for mapping file drives over NETBIOS or TCP/IP, supported by Microsoft for Windows-based file serving.

Single Level Store (SLS) The single address space shared by all PowerAS architecture programs running in all processes. SLS provides a mapping for all storage (memory and disk) in an AS/400 system.

SLIC See *System Licensed Internal Code*.

SLS See *Single Level Store*.

SMB See *Simple Message Block*.

Supply Chain Management (SCM) Generally used for the business applications that are used to manage the supply chain.

System Licensed Internal Code (SLIC) Contains all kernel (privileged) code for AS/400 systems, plus a large amount of low-level user runtime code that does not require privilege.

Technology Independent Machine Interface (TIMI) See *Machine Interface*.

UNIX An operating system developed at Bell Laboratories (trademark of UNIX System Laboratories, licensed exclusively by X/Open Company, Ltd.).

Index

Symbols

#pragma 113, 116
_CVTSPP 111
_CVTSPP() function 136
  error conditions 137
  parameters 137
  return value 137
  syntax 136
_ILECALL 111
_ILECALL() function 129
  error conditions 133
  parameters 130
  restrictions 133
  return value 133
  syntax 129
_ILELOAD 111, 112
_ILELOAD() function 125
  authorities and locks 125
  error conditions 127
  parameters 125
  restrictions 126
  return value 126
  syntax 125
_ILESYM 111, 112
_ILESYM() function 127
  authorities and locks 127
  error conditions 129
  parameters 128
  restrictions 128
  return value 128
  syntax 127
_MEMCPY_WT() and _MEMCPY_WT2() functions 134
  error conditions 135
  parameters 135
  return value 135
  syntax 134
_SETCCSID 111
_SETCCSID() function 137
  parameters 137
  return value 138
  syntax 137
_SETSPP 111
_SETSPP() function 135
  error conditions 136
  parameters 136
  return value 136
  syntax 136

A

ABI 9
accessing files 60
accessing files from AS/400 PASE 60
activation group 109
address resolution 42
addressing 40
AIX 3
AIX exploitation points 186
API 24
  analysis 45
  analysis tool 45
  restrictions 24
API analysis report 47
Application Binary Interface (ABI) 3
Application Factory 12, 205
argument list 110
AS/400 architecture 39
AS/400 DDS 118
AS/400 Interactive Debugger 178
AS/400 PASE
  advantages of porting to 11
  application debugging 84
  C code 111
  capabilities 11
  codesets 198
  development environment 10
  devices 23
  functions 111
  functions for accessing ILE 111
  getting started 33
  globalization 187
  hardware considerations 10
  ILE comparison 211
  limitations 11, 23
  locales 193
  performance 10
  planning a port to 6
  porting examples 147
  positioning 4
  runtime environment 1, 3
  shells 19
  starting an application 65
  unsupported system calls 85

user profiles 185
using dbx 84
utilities 19
AS/400 performance 209
AS/400 Portable Application Solutions Environment (AS/400 PASE) 3
AS/400 workloads 209
AS/400e 5, 10
authority management 42

B

beginning a port 57
buffered versus unbuffered I/O 44
build 6
build_ILEarglist 111
build_ILEarglist() 121
build_ILEarglist() function 121
parameters 122
restrictions 124
return value 125
syntax 122

C

call outs to ILE from AS/400 PASE 108
calling AS/400 PASE from ILE 141
calling ILE from AS/400 PASE 111
calling Java from AS/400 PASE 108
capabilities 14
CCSID 23, 24, 71, 87, 108, 137
character encoding conversion support 190
character sets 43
checking system support 35
coding 23
compilation 118
compilation and runtime commands 118
compiler options 112
compiling applications on AIX 58
computationally intensive applications 6
configuration tips 62
context 42
creating your AS/400 solution 36
CRTCMOD 118, 180
CRTPF 118
CRTSRVPGM 118

D

data encoding considerations 87
database access 210
database porting 87
database support 191
date and time services 191
DB2 Universal Database for AS/400 5
DB2CLI 87
dbx 180
dbx debugger utility 84
debugging 84
development environment 107
device 23
device support 192

E

EBCDIC versus ASCII 43
Enterprise Resource Planning (ERP) 3
exec 6
exec() 42
exit status 22

F

file system 23, 104
file system support 188
file systems 14, 23
finding messages on the AIX system 179
finding messages on the AS/400 system 179
flags 20
fork 6
fork() 41, 42

G

general PASE instructions 34
general system set up 35
GNU perl 159
GNU zip 147

H

header and export files 58
header files 109

I

I/O controllers versus device drivers 44
IBM’s strategy 4
ILE integration 99
Independant Service Providers (ISVs) 11
installation and packaging 207
integrated file system 14

Integrated Language Environment (ILE) 39
integrated language environment (ILE) 3, 8
Integrated Netfinity Server 4
Interactive Debugger 178
interactive terminal support 189
internationalization 209
IOCTL support 31

J
job logs 174
job structure 41

L
leveraging existing skills 11
libraries 24
library 24
library resolution 42
license types
  GNU public license (GPL) 12
  Library GNU public license (LGPL) 12
licensing 34, 207
licensing issues 33
limitations 23
Linux 4
locale support 187
logging facilities 22

M
Memory model 102
memory model 102
MI program template 100
Midrange-L 147, 223

N
national language 209

O
object oriented architecture 39
online transaction processing (OLTP) 3
OpenDX 147, 150
operations 208
OS/400 access 22
OS/400 FTP client 61
OS/400 FTP daemon 61

P
PartnerWorld for Developers 5, 223
PASE 61
PASE_LANG 87
PASE_SYSCALL_NOSIGILL 85
performance 107
porting and testing 119
porting mechanism 57
porting your modules 119
positioning AS/400 PASE 4
possible errors returned 77
preparing the application 35
printing 24, 210
problem determination and messages 173
problem determination tools 173
problem management 210
process structure 41
program model 102
programming resources 203
public license
  Nethack GNU public license (NGPL) 12

Q
QIBM_PASE_CCSID 87
QIBM_PASE_DESCRIPTOR_STDIO 68
QIBM_USE_DESCRIPTOR_STDIO 68, 104
QOpenSys 42
Qp2CallPase 79
  authorities and locks 79
  information 141
  invocation from an ILE application 79
  parameters 80
  programming notes 81
  restrictions 81
  return value 81
  syntax 79
Qp2RunPase 71
  authorities and locks 73
  information 141
  invocation from an ILE application 71
  parameters 73
  programming notes 76
  syntax 72
QP2SHELL
  error conditions 69
  invocation from a 5250 screen 65
  parameters 66
  programming notes 68

syntax 66
Qp2SignalPase 82
calling a procedure 82
parameter 82
programming notes 83
restrictions 83
return value 83
syntax 82
QP2TERM
invocation from the AS/400 PASE terminal 69
parameters 70
programming notes 71
syntax 70
Qp2TERM 69

R

recommendations 34
remote file systems 62
requirements 33
result type 110
ROUND_QUAD 110, 112
ROUND_QUAD function 110
RS/6000 5
runtime commands 118
runtime modes 7
runtime support 105
runtime symbols 19

S

security 209
security considerations 185
Segment Lookaside Buffer (SLB) 10
services 6
setting up variables and structures 109
shared addressing 100
shared libraries 14
shells 19, 24, 192
SIGILL 85, 86
illegal instruction 57
unsupported instruction 85
signature 110
SIGTERM 103, 108
single level store (SLS) 100
size_ILEarglist 111
size_ILEarglist() function 120
parameters 120
return value 121
syntax 120
SLIC trace facilities 57
SLS (single level store) 9, 100
SMB 62
sockets support 104, 188
spawn 6
spawn() 42
stderr 189
stdin 189
stdout 189
storage management 40
successfully ported applications 12
switching to AIX 161
switching to the AS/400 system 162
syntax 20
System Licensed Internal Code (SLIC) 8, 39
system structure 101
systemCL 111
systemCL function 138
authorities and locks 138
error codes 140
parameters 138
return value 140
syntax 138

T

TERASPACE 103
terminal I/O 43
testing 6
testing the application 35
testing your modules 119
time to market 11
TIMI 9
tools on AIX 173
tools on OS/400 173

U

UNIX perspective 39
unrecognized APIs 49
unsupported runtime library members 16
unsupported system calls 31, 85
user profiles 42
user-defined APIs 49
using FTP 60
utilities 192

V

viewing AS/400 PASE programs 169

VLOG 85, 175

W

work management 169
WRKACTJOB 177

X

XCOFF 101
xlc 118
X-Windows support 192

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at **ibm.com**/redbooks
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-5970-00 Porting UNIX Applications Using AS/400 PASE
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. **ibm.com**/privacy/yourprivacy/

Porting UNIX Applications Using AS/400 PASE

(0.2”spine)
0.17”<->0.473”
90<->249 pages

Porting UNIX Applications

Using AS/400 PASE

Explores the Portable Application Solutions Environment

Offers links to download UNIX application examples

The porting guide for beginners and experts

This redbook positions the new AS/400 Portable Application Solutions Environment (PASE), referred to as AS/400 PASE, with the other options available to AS/400 application developers. It helps you determine whether an AIX-based application or module can be easily ported to PASE, or whether you should consider an ILE port.

There are five primary audiences for this redbook. These audiences include executives and planners who are looking at using AS/400 PASE for portions of their solutions; UNIX application providers who are considering a port of their application to the AS/400 platform; businesses that already leverage both AS/400 and AIX solutions and want to consolidate their solutions into one platform; AS/400 shops looking to purchase or use an AS/400 PASE application; and AS/400 shops looking to purchase a UNIX solution who want to encourage the vendor of that solution to consider AS/400 delivery and support.

With these audiences in mind, we divided this book in three parts. Part 1, “The AS/400 PASE runtime environment”, applies to all audiences and provides an overview of the product. Part 2, “AS/400 PASE from a UNIX perspective”, targets UNIX application providers and businesses that are looking to port their own in-house applications from AIX to the AS/400 system. Part 3, “AS/400 PASE from an AS/400 perspective”, is intended for all technical audiences, including AS/400 system operators.

SG24-5970-00 ISBN 0738417971

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks