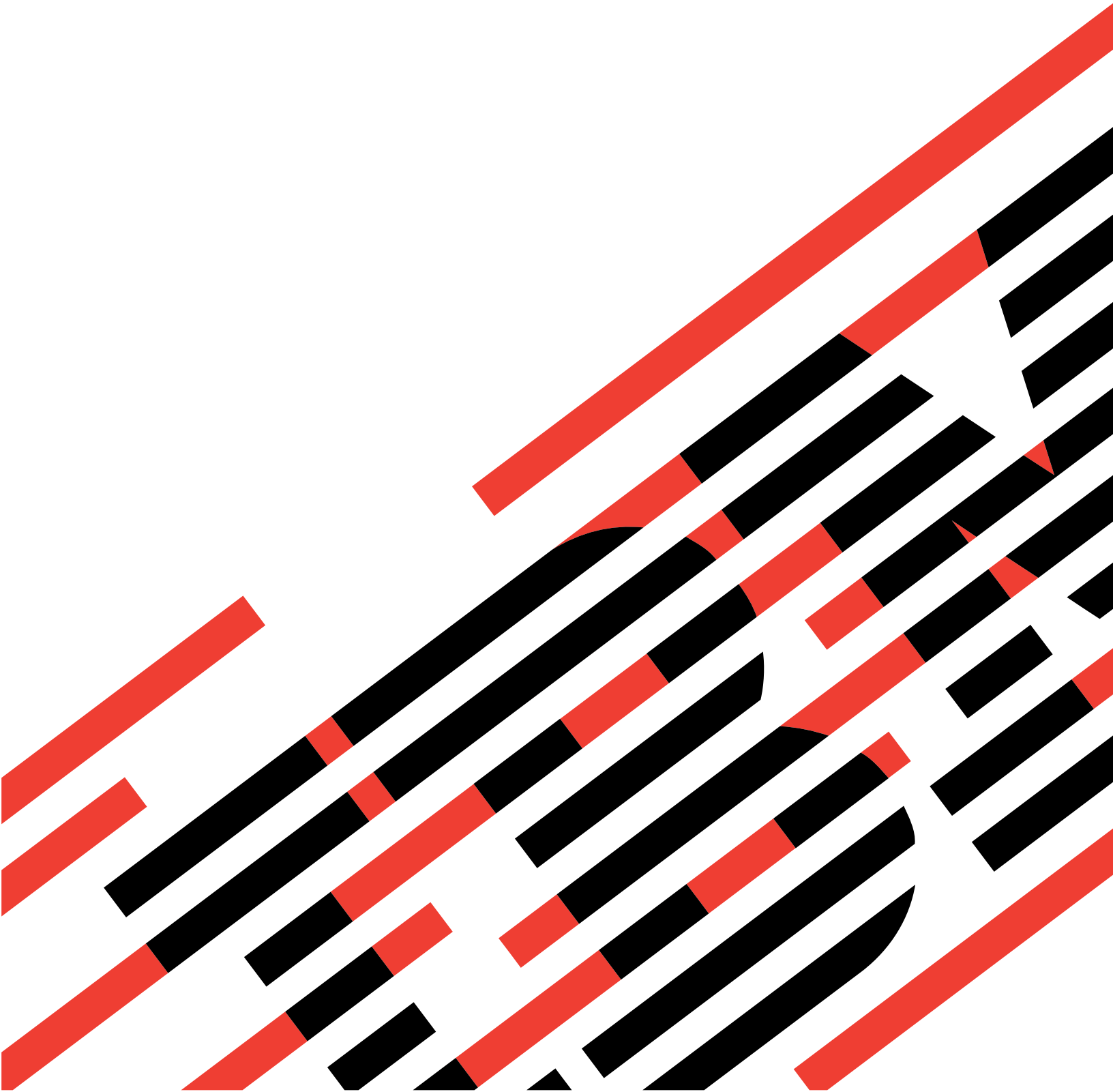
IBM

eserver

iSeries

OS/400 PASE



IBM

@server

iSeries

OS/400 PASE

目录

OS/400 PASE

“OS/400(R) 可移植应用程序解决方案环境”（OS/400 PASE）允许将 AIX(R) 应用程序方便地移植到 iSeries(TM) 服务器。OS/400 PASE 提供了集成的运行时环境，使您可以运行所选的 UNIX (R) 应用程序，而无须进行复杂的 UNIX 系统管理。 » OS/400 PASE 还提供工业标准和实际标准外壳程序以及提供强大脚本编制环境的实用程序。 «

要想更多了解 OS/400 PASE，参见以下内容。也可以在此发行版中查找关于『V5R2 的新增功能』和如何第 3 页的『打印此主题』的信息。

V5R2 的新增功能

以下是 V5R2 中对于 OS/400 PASE 的一些重大功能增强和更改:

- OS/400 PASE 支持在 OS/400 PASE 环境中安装两个 AIX 编译器产品中的任何一个，从而消除了在 AIX 上测试和编译 OS/400 PASE 程序的需求。有关详细信息，参见第 9 页的『编译 AIX 源』。
- 伪终端（PTY）支持和 UNIX 风格的作业控制。有关详细信息，参见第 41 页的『伪终端（PTY）』。
- 100 多个新的实用程序。要获得完整的列表，参见 OS/400 PASE 外壳程序和实用程序主题。
- 添加了以下新库。要获得完整的列表，参见 OS/400 PASE 运行时库主题。

libcur.a	AIX 旧的 Curses 库
libg.a	调试支持
libgair4.a	内部 X Windows 支持
libl.a	lex 支持

libld.a	目标文件访问例程库
libm.a	IEEE 数学库
libPW.a	程序工作台库
libxcurses.a	Curses 库
libXi.a	X Windows 输入处理
libXtst.a	X Windows 测试支持
liby.a	yacc 支持

- 对于 OS/400 PASE 程序使用的任何不受支持的系统调用，作业记录中会出现新消息（MCH3204）。此消息的文本包括系统调用的名称和导致错误的指令地址。
- 新增的和已更改的 OS/400 PASE 运行时函数:
  - _CVTERRNO（将 OS/400 PASE errno 转换为 ILE errno）
  - _ILECALLX（增强了的 ILE 过程调用）
  - _PMGCALL（调用 OS/400 程序）
  - _RETURN（不退出 OS/400 PASE 而返回）
  - _RSLOBJ 和 _RSLOBJ2（解析至 OS/400 对象）
  - _STRLEN_SPP 和 _STRCPY_SPP（使用 16 字节 ILE 指针的字符串处理）
  - Qp2paseCCSID（检索 OS/400 PASE CCSID）
  - Qp2jobCCSID（从上一次 OS/400 PASE CCSID 的设置中检索作业缺省 CCSID）
  - faccessx
  - fchdir
  - fclear
  - fclear
  - getaddrinfo 和 getnameinfo
  - getcontext 和 setcontext
  - getpri、getpriority 和 setpriority
  - getprocs64 和 getthrds64
  - gettimer 和 settimer
  - msem_init、msem_lock、msleep、msem_unlock 和 msem_remove
  - pread 和 pwrite
  - setgroups
  - sigstack 和 sigaltstack（备用信号堆栈）
  - statpriv
  - statvfs 和 fstatvfs
  - sync
  - ustat
- 新增的和已更改的用于 OS/400 PASE 的 (ILE) API:
  - QP2SHELL2（与 QP2SHELL 相似，但在调用程序的激活组中运行）
  - Qp2ptrsize（检索 OS/400 PASE 指针大小）
  - Qp2paseCCSID（检索 OS/400 PASE CCSID）
  - Qp2jobCCSID（检索 OS/400 PASE CCSID 上一次设置时的作业缺省值 CCSID）

- Qp2errnop（为当前线程定位 OS/400 PASE errno）
- Qp2malloc（分配 OS/400 PASE 堆内存）
- Qp2free（释放 OS/400 PASE 堆内存）
- Qp2dlopen（动态装入 OS/400 PASE 模块）
- Qp2dlsym（在 OS/400 PASE dlopen 打开的模块中查找符号）
- Qp2dlclose（关闭和卸装由 OS/400 PASE dlopen 装入的模块）
- Qp2dlerror（为最后动态装入的操作检索错误信息）
- 对用于按地址的自变量和结果以及在不是由 OS/400 PASE 启动的线程中调用 OS/400 PASE 过程的 Qp2CallPase（和 Qp2CallPase2）进行了增强

- OS/400 PASE 语言环境（和其它支持全球化的文件）现在和 OS/400 语言功能代码封装在一起；有关详细信息，参见第 39 页的『全球化』。另外，还提供 200 多个供 X Windows 处理不同的键盘和字符集的新文件，以及以下 65 个新语言环境。要获得完整列表，参见 OS/400 PASE 语言环境主题。

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

如何查看新增的或已更改的功能

为了帮助您查看何处作了技术更改，此信息使用:

- » 标记新增的或已更改的信息开始处的图像。
- « 标记新增的或已更改的信息结束处的图像。

要查找其它有关本发行版中新增的或已更改的功能的信息，参见《用户备忘录》  。

打印此主题

要查看或下载 PDF 版本，请选择 OS/400 PASE 信息（大约 211 KB 或 52 页）。

保存 PDF 文件

要在工作站上保存 PDF 以便查看或打印:

1. 在浏览器中用右键单击 PDF（右键单击上面的链接）。
2. 单击**目标另存为...**

3. 浏览至要在其中保存 PDF 的目录。
4. 单击**保存**。

下载 Adobe Acrobat Reader

如果需要 Adobe Acrobat Reader 来查看或打印这些 PDF，您可以从 Adobe 站点（www.adobe.com/products/acrobat/readstep.html）  下载。

OS/400 PASE 入门

跨平台应用程序开发和部署是任何有效的商务计算环境的关键组成部分。同样重要的是使用的简易性和系统提供功能的集成 — iSeries 和 AS/400e(TM) 服务器的标志。由于业务转移到日益开放的计算环境之中，您很有可能会发现，归档这些离散的目标很困难、费时和昂贵。例如，您可能想要熟悉在 AIX 操作系统上运行的应用程序，并充分利用 AIX 操作系统的功能，但您却不想增加管理 AIX 和 OS/400 操作系统的负担。

此时 OS/400 可移植应用程序解决方案环境（OS/400 PASE）可帮助您解决问题。OS/400 PASE 允许您在 OS/400 上稍加更改或不作任何更改的情况下运行许多 AIX 应用程序二进制，并有效扩展您的平台解决方案任务夹。

要学习更多有关 OS/400 PASE 的内容，参见以下主题:

- OS/400 PASE 是什么？
- OS/400 PASE 何时是应用程序开发的有用选项？
- 安装 OS/400 PASE

OS/400 PASE 是什么?

“OS/400 可移植应用程序解决方案环境”（OS/400 PASE）是用于运行在 OS/400 上的 AIX 应用程序的集成运行时环境。它支持 AIX 的“应用程序二进制接口”（ABI），并支持 AIX 共享库、外壳程序和实用程序所支持功能中的大部分。OS/400 PASE 支持 PowerPC(TM) 机器指令的直接执行，因此它没有只模仿机器指令的环境的缺点。

OS/400 PASE 应用程序有以下特点:

- 可以用 C、C++、Fortran 或 PowerPC 汇编程序来编写
- 使用与 AIX PowerPC 应用程序相同的二进制可执行文件格式
- 在 OS/400 作业中运行
- 使用 OS/400 系统功能，如文件系统、安全性和套接字

请记住，OS/400 PASE 不是在 OS/400 上的 UNIX 操作系统。OS/400 PASE 的设计目的是为了作很少更改或不作更改便能在 OS/400 上运行 AIX。来自任何其它基于 UNIX 环境中的程序要先被改写才能在 AIX 上进行编译，这是在 OS/400 PASE 中运行这些程序的第一步。

OS/400 PASE 集成的运行时在 iSeries 服务器上的“许可内码”（LIC）内核上运行。该系统集成了跨越 OS/400 PASE 和其它运行时环境（包括 ILE 和 Java(TM)）的许多常用 OS/400 功能。OS/400 PASE 实现了大部分的 AIX 系统调用。 » 通过控制 OS/400 PASE 能访问的存储器并限制它只能使用无特权的机器指令，OS/400 PASE 的系统支持加强了系统安全性和完整性。 «

无须费力即可迅速部署应用程序

在很多情况下，可能作很少更改或不作更改就能在 OS/400 PASE 中运行 AIX 程序。所需的 AIX 编程技巧水平会因 AIX 程序的设计而异。另外，通过在程序设计上提供额外的 OS/400 应用程序集成（如使用 CL 命令），便无须对应用程序用户的配置过多担忧。

OS/400 PASE 为想在 OS/400 市场上共享成功的解决方案开发者增加了另一个移植选择。通过提供可显著减少移植时间的方法，OS/400 PASE 能使解决方案的开发者加速推出产品并增加投资回报。

OS/400 上的 AIX 技术的广泛子集

OS/400 PASE 实现了以 AIX 技术的广泛子集为基础的应用程序运行时，其中包括：

- 标准 C 和 C++ 运行时（线程安全和非线程安全）
- Fortran 运行时（线程安全和非线程安全）
- `pthreads` 线程包
- 用于数据转换的 `iconv` 服务
- Berkeley 软件分发（BSD）等量支持
- 对 Motif 小配件集合的 X-windows 客户机支持
- 伪终端（PTY）支持

应用程序在 AIX 工作站上开发和编译，然后在 OS/400 上运行，而该 AIX 工作站运行的是与 OS/400 PASE 支持的某级 AIX 兼容的 AIX 级别。

另一个选择是，您可以在 OS/400 PASE 环境中安装以下两个产品中的一个，以完全在 OS/400 PASE 中开发、编译、构建和运行应用程序。

- IBM(R) VisualAge C++ Professional for AIX（V6）
- IBM C for AIX（V6）

有关详细信息，参见第 9 页的『编译 AIX 源』。

OS/400 PASE 还包含 Korn、Bourne 和 C 外壳程序，以及大约 200 个提供功能强大的脚本编制环境的实用程序。有关更多信息，参见 OS/400 PASE 外壳程序和实用程序。

OS/400 PASE 使用 IBM 在 AIX 和 OS/400 操作系统的通用处理器技术方面的投资。为在 OS/400 PASE 运行时中运行应用程序，PowerPC 处理器从 OS/400 方式切换至 AIX 方式。

在 OS/400 PASE 中运行的应用程序与 OS/400 集成文件系统和 DB2(R) 通用数据库(R) iSeries 版集成在一起。它们可以调用 Java 和“集成语言环境”（ILE）应用程序，也可以被这些应用程序调用。一般而言，它们可以利用 OS/400 操作环境的所有功能和特性，如安全性、消息传递、通信、备份和恢复。同时，它们可以利用派生自 AIX 接口的应用程序接口。

OS/400 PASE 对于应用程序的开发何时是有用的选择？

在决定如何将 AIX 应用程序移植至 iSeries 服务器时，OS/400 PASE 提供很高的灵活性。当然，OS/400 PASE 只是您可以选择的多个选项之一。

API 分析

在确定应用程序是否适合 OS/400 PASE 时，首先要做的是应用程序分析，包括分析应用程序使用的 API 、库和实用程序及应用程序在 OS/400 上运行的效率。IBM PartnerWorld(TM)  组使用 API Analysis Tool 

提供此区域的帮助。API 分析工具是一个免费的移植评估工具，用于分析应用程序和描述潜在的障碍性阻塞。有关该分析工具如何适用于将应用程序移植至 OS/400 PASE 的过程的更多信息，参见第 8 页的『准备在 OS/400 PASE 中运行的程序』。

潜在的 OS/400 PASE 应用程序的特征

以下是一些有用的指导，您可以在决定是否使用 OS/400 PASE 时考虑使用这些指导:

- **AIX 应用程序是否为高度计算密集型的程序?**
  OS/400 PASE 通过提供高度优化的数学库为在 iSeries 服务器上运行计算密集型的应用程序提供了一个良好的环境。
- **» 应用程序是否非常依赖于只在 OS/400 PASE 中支持（或只在 ILE 中部分支持）的函数（如 fork()、X-Windows 或伪终端（PTY））? «**
  OS/400 PASE 提供对 fork()和 exec() 的支持，而当前在 OS/400 系统中并未提供这种支持（除了通过 spawn() 之外，通过它可将 fork() 函数与 exec() 函数结合起来)。
- **应用程序是使用基于 AIX 的复杂构件过程还是使用测试环境?**
  OS/400 PASE 允许您使用基于 AIX 的构件过程。当存在着未准备好传送至新平台的现有的复杂过程时，该过程特别有用。
- **应用程序是否与 ASCII 字符集有相关性?**
  OS/400 PASE 为具有这些需求的应用程序提供了良好的支持。
- **» 应用程序是否会进行大量的指针处理? 还是会将整数转换（cast）为指针?**
  OS/400 PASE 以很低的性能成本和将整数转换为指针的能力支持 32 位和 64 位的 AIX 寻址模型。«

当 OS/400 PASE 可能是最好的解决方案时

对于提供大量必须从 ILE 调用的可调用接口并且具有以下任一特征的代码，OS/400 PASE 通常不是一个好选择:

- » 需要更高性能的调用和返回的代码，此代码所需的这些性能要比在一个已激活的 OS/400 PASE 程序中每次调用 OS/400 PASE 过程（使用 Qp2CallPase API）时启动或结束 OS/400 PASE 所提供的性能高。 «
- 需要在 ILE 调用程序和库代码之间共享内存或名称空间的代码。OS/400 PASE 程序不会与调用它的 ILE 代码隐式分享内存或名称空间。（但是，从 OS/400 PASE 中调用的 ILE 代码可以分享或者使用 OS/400 PASE 的内存。）

安装 OS/400 PASE

» 所有 iSeries 服务器均可免费获得 OS/400 PASE。建议安装 OS/400 PASE; 有些系统软件（如增强的域名服务器（DNS）和 ILE C++ 编译器）需要 OS/400 PASE 支持。 «

在服务器上安装 OS/400 PASE:

1. 在 OS/400 命令行输入 GO LICPGM。
2. 选择 11. 安装许可程序。
3. 选择选项 33（5722SS1 — 可移植应用程序解决方案环境）。

» 语言环境安装

OS/400 PASE 产品只安装与已安装在 OS/400 上的语言功能部件有关的语言环境对象。如果需要服务器上的语言功能部件之外的语言环境，则可能需要订购和安装额外的 OS/400 语言功能部件。更多信息，参见 第 39 页的『全球化』和 OS/400 PASE 语言环境。 «

将应用程序移植到 OS/400 PASE 的软件开发者的许可注释:

OS/400 PASE 提供 OS/400 系统上的 AIX 运行时库的子集。OS/400 许可证授权您使用随 OS/400 所交付的任何库代码。此许可证不暗示不随 OS/400 PASE 交付的 AIX 库的许可证。所有 AIX 产品由 IBM 单独许可。

开始将您自己的应用程序移植到 OS/400 PASE 时，可能会发现应用程序具有与不是随 OS/400 PASE 交付的 AIX 库的相关性。将这些库移植到 OS/400 系统之前，应该确定哪些软件产品提供这些库，并检查此软件产品的许可证协议条款和条件。它可能是使用 IBM 或第三方来将附加中间件的相关性移植到 OS/400 系统中所必需的。在开始移植之前，应该调查涉及您正在移植的代码的每份许可证发放协议。如果需要在适当的位置找到与您认为属于 IBM 的库有关的许可证协议，请与 IBM 销售代表、IBM 移植中心之一、罗彻斯特的客户技术中心（Custom Technology Center）或 PartnerWorld for Developers 联系。

规划 OS/400 PASE

OS/400 PASE 提供了 OS/400 中的 AIX 运行时环境，使您可以毫不费力地将 AIX 应用程序移植至 iSeries 服务器。实际上，多数 AIX 程序无须更改即可在 OS/400 PASE 中运行。这是因为 OS/400 PASE 提供了多数与 AIX 中的共享库相同的共享库，并提供了大量的 AIX 实用程序子集，这些子集以与在 pSeries(TM) AIX PowerPC 处理器上相同的运行方式在 iSeries PowerPC 处理器上直接运行。

在开始运行 OS/400 PASE 之前，您需要考虑以下几点:

- **AIX 二进制目标发行版和将运行二进制的 OS/400 PASE 的发行版之间存在相关性。**

  如果在 AIX 上编译 OS/400 PASE 应用程序，则在 AIX 中创建的应用程序二进制需要与要运行应用程序的 OS/400 PASE 的版本兼容。下表显示了与 OS/400 PASE 的不同版本兼容的 AIX 二进制版本。例如，为 AIX R4.3 创建的 32 位应用程序将在 OS/400 PASE V4R5、V5R1 或 V5R2 上运行，但不会在 OS/400 PASE V4R4 上运行。类似地，为 AIX R4.3 创建的 64 位应用程序将在 OS/400 PASE V5R1 上运行，但不会在 OS/400 PASE V4R4、V4R5 或 V5R2 上运行。

AIX 发行版	OS/400 V4R4	OS/400 V4R5	OS/400 V5R1	OS/400 V5R2
4.2（32 位）	X	X	X	X
4.3（32 位）	-	X	X	X
4.3（64 位）	-	-	X	-
5.1（32 或 64 位）	-	-	-	X（参见注）

  注: 64 位 OS/400 PASE V5R2 应用程序必须在 AIX 5L(TM) R5.1 上重新编译。有关详细信息，参见第 8 页的『准备在 OS/400 PASE 中运行的程序』。

- **OS/400 PASE 未在 OS/400 中提供 AIX 内核。**

  相反，共享库所需的任何低级系统函数会被传送到 OS/400 内核或集成的 OS/400 函数。在这点上，OS/400 PASE 在 AIX 和 OS/400 平台间建立了联系: 您的代码使用与 AIX 上共享库中的 API 相同的语法，但 OS/400 PASE 程序在 OS/400 作业中运行并由 OS/400 管理，就象任何其它 OS/400 作业一样。

- **在大多数情况下，在 OS/400 PASE 中调用的 API 的运行情况与它们在 AIX 中的运行情况完全相同。**

  但是，某些 API 在 OS/400 PASE 中的运行情况可能有所不同，或者在 OS/400 PASE 中不受支持。因此，您第 8 页的『准备在 OS/400 PASE 中运行的程序』的规划应从使用 API 分析工具进行详尽的代码分析开始。此工具为您提供了在将 AIX 应用程序移植至 OS/400 PASE 时需要考虑的程序修改类型的综合摘要。

- **考虑 AIX 和 OS/400 平台之间存在的某些差异:**

  – 在通常情况下，AIX 第 12 页的『区分大小写』，但某些 OS/400 文件系统则不会。

– AIX 通常使用 ASCII 进行第 38 页的『数据编码』，而 OS/400 则通常使用 EBCDIC。如果要管理从 OS/400 PASE 程序中调用集成语言环境（ILE）代码的详细信息，您需要注意这一点。例如，在执行由 OS/400 PASE 对任意 ILE 过程的调用时，您必须显式地编写 OS/400 PASE 程序，以便处理对字符串的字符编码转换。OS/400 PASE 运行时支持包括用于字符编码转换的 `iconv_open()`、`iconv()` 和 `iconv_close()`函数。

  注: OS/400 PASE 和 ILE 均可独立实现 `iconv()` 接口，并各自拥有自己的转换表。由于 OS/400 PASE `iconv()` 支持所支持的转换是以字节流文件的形式存储于集成文件系统中的，因此用户可以对它们进行修改和扩展。
– AIX 应用程序要求行（例如，在文件和外壳程序脚本中）以换行（LF）结束，但个人计算机（PC）软件和 OS/400 软件则典型地以第 13 页的『集成文件系统文件中的行终止字符』结束行。
– 您在 AIX 上使用的某些脚本和程序可能使用标准实用程序的硬编码路径，因此您需要修改路径，以便反映将在 OS/400 PASE 中使用的路径。有关更多信息，参见第 9 页的『分析程序与 OS/400 PASE 的兼容性』。

OS/400 PASE 会自动处理某些问题。例如，尽管通常情况下不对读取或写入文件描述符（字节流文件或套接字）的数据执行转换，但在使用系统提供的 OS/400 PASE 运行时服务（包括任何系统调用或随 OS/400 选项 33 交付的共享库中的运行时函数）时，OS/400 PASE 将根据需要执行 ASCII 至 EBCDIC 的转换。

您可以使用其它低级函数（如 _ILECALL），调用 ILE 函数和 API 来扩展 OS/400 PASE 程序的功能，但如上所述，您可能需要处理数据转换。此外，将这些扩展编入程序将需要使用附加的第 13 页的『复制头文件』和第 14 页的『复制导出文件』文件。

准备在 OS/400 PASE 中运行的程序

准备在 OS/400 上有效运行的 AIX 程序所需采取的步骤随程序的性质以及是否需要使用 OS/400 专用的接口和函数而不同。

如果尝试将 UNIX 应用程序移植至 OS/400 PASE，您必须首先确保应用程序将使用 » AIX 编译器 « 进行编译。在某些情况下，您需要修改 UNIX 程序来达到此要求。

» **注**: OS/400 PASE V5R2 支持 AIX 5L R5.1。想要在 OS/400 PASE 中运行的新 64 位应用程序必须在 AIX 5L R5.1 上编译，并且必须重新编译现有的 64 位应用程序。OS/400 PASE 支持 AIX 先前发行版中的 32 位应用程序，而无须重新编译。 «

第 1 步: 第 9 页的『分析程序与 OS/400 PASE 的兼容性』
建议在所有情况下均执行此过程中的第一步。使用 API 分析工具来获得有关程序使用的 API 的全面报告，以及您希望在 OS/400 PASE 中执行这些 API 的方式。

第 2 步: 第 9 页的『编译 AIX 源』
在确定您的程序适合作为 OS/400 PASE 程序，并且完成要在 OS/400 PASE 中运行可能需要进行的更改后，请编译源程序。（如果 AIX 程序的分析显示无须更改即可在 OS/400 PASE 中运行，您无须重新编译程序。）

» 使用 AIX 系统来编译 OS/400 PASE 程序，或者可以选择性地在 OS/400 PASE 中安装两个 AIX 编译器产品的其中之一，以便在 OS/400 PASE 环境中编译程序。 «

第 3 步: 第 11 页的『将 OS/400 PASE 程序复制到 iSeries 服务器』
» 如果在 AIX 系统中编译了 OS/400 PASE 程序，请将二进制文件复制至 iSeries 服务器。 «

第 4 步: （可选）第 13 页的『定制 OS/400 PASE 程序来使用 OS/400 功能』

如果要定制 AIX 应用程序以使用 OS/400 专用的接口，» 并且在 AIX 上编译应用程序 «，在编译 OS/400 PASE 程序之前，您必须将一个或多个 OS/400 头文件或导出文件复制至 AIX 系统。

分析程序与 OS/400 PASE 的兼容性

评估 UNIX C 应用程序移植到 iSeries 服务器的可移植性的第一步包含分析应用程序中使用的接口。此 API 分析标识了那些在非业界标准和 OS/400 上不支持的应用程序内使用的接口。由于 OS/400 机器与 UNIX 机器的体系结构不同，它还标识符合标准但所受支持不同的接口。

API Analysis Tool  包含前端和后端进程。前端进程扫描已编译的应用程序以抽取应用程序所使用的接口（外部函数和数据），并生成所有这些接口的列表。后端进程将此接口列表作为输入，并将接口与典型系统 API 及其支持的数据库相比较。

API 分析工具的前端进程是 UNIX 外壳程序脚本。它使用 `nm` 或 `dump` 命令来从应用程序的外部符号表中查找符号信息。

剥离了符号的二进制可以包含要分析的工具的足够动态绑定信息。静态绑定的二进制将库接口从分析中除去，但仍可暴露系统调用相关性以用于分析。

在编译之前要执行的附加分析

除了从 API 分析工具中收集的信息外，还应该收集下列信息:

- **获得应用程序所使用的库列表**: 分析工具提供关于应用程序使用的一些标准 API 的反馈，但它不会寻找许多常见 API 集。库分析可帮助标识应用程序使用的一些中间件 API。可以针对每个命令和共享的对象运行下列命令，来获得应用程序所需的库列表:

  ```
  dump -H binary_name
  ```

- » **检查硬编码路径名的代码**: 如果运行更改凭证的程序或者甚至要在 OS/400 PASE 环境变量 PASE_EXEC_QOPENSYS=N 时运行程序或脚本，则可能需要更改硬编码路径名。

  因为 `/usr/bin/ksh` 是绝对路径（从根目录开始），如果没有找到它或者它不是字节流文件，则 OS/400 PASE 自动搜索 /QOpenSys 文件系统以查找路径名 `/QOpenSys/usr/bin/ksh`。QShell 实用程序不是字节流文件，所以 OS/400 PASE 会搜索 /QOpenSys 文件系统，甚至当原始（绝对）路径是到 QShell 实用程序（如 `/usr/bin/sh`）的符号链接时也是如此。«

编译 AIX 源

当程序只使用 AIX 接口时，您使用所有必需的 AIX 头编译并与 AIX 库链接，来为 OS/400 PASE 准备二进制。请记住，OS/400 PASE 不支持与 AIX 系统提供的共享库静态绑定的应用程序。

OS/400 PASE 程序在结构上与用于 PowerPC 的 AIX 程序相同。

» OS/400 PASE 选项 33 不包括编译器。可使用 AIX 系统来编译 OS/400 PASE 程序，或在 OS/400 PASE 中选择安装两个 AIX 编译器产品之一，以便在 OS/400 PASE 环境中编译程序。 «

在 pSeries 服务器上使用 AIX 编译器

您可以构建使用任何 AIX 编译器和链接程序（可生成与用于 PowerPC 的 AIX 应用程序二进制接口（ABI）兼容的输出）的 OS/400 PASE 程序。OS/400 PASE 为使用不存在于 PowerPC 中的 POWER 结构指令（高速缓存管理 POWER 指令除外）的二进制提供指令仿真支持。

在 OS/400 PASE 中使用 AIX 编译器

» 在 OS/400 PASE 环境中，OS/400 PASE 支持安装以下单独提供的任何一个 AIX 编译器:

- IBM VisualAge C++ Professional for AIX，V6（5765-F56）（本产品包括 IBM C for AIX 编译器。）
- IBM C for AIX，V6（5765-F57）

使用这些产品，可以完全在 iSeries 服务器上的 OS/400 PASE 环境内开发、编译、构建和运行 AIX 应用程序。

有关订购和安装这些产品的更多信息，参见 Application Factory Web 站点上的 VisualAge C++ for AIX compiler installation in OS/400 PASE 页。 «

开发工具

» 在 AIX 上使用的许多开发工具（例如，ld、ar、make 和 yacc）包括在 OS/400 PASE 内。有关详细信息，参见 OS/400 PASE 外壳程序和实用程序主题。许多来自其它源的 AIX 工具（例如，开放源工具 gcc）也可以在 OS/400 PASE 中工作。

iSeries Tools for Developers PRPQ（5799-PTL）也包含大量工具来帮助开发、构建和移植 iSeries 应用程序。有关此 PRPQ 的更多信息，参见 Application Factory - iSeries Tools for Development Web 站点。 «

适用于处理指针的编译器注释

- xlc 编译器通过使用 -qlngdbl128 和 -qalign=natural 的组合，为 16 字节对齐（适用于长双精度类型）提供有限支持。ILEpointer 类型需要这些编译器选项来确保 MI 指针在结构内是 16 字节对齐的。使用选项 -qldbl128 将长双精度类型强制为 128 位类型（要求使用 libc128.a 来为长双精度字段处理诸如 printf 的运算）。

  获得选项 -qlngdbl128 并与 libc128.a 链接的简易方法是使用 xlc128 命令，而不是 xlc 命令。
- 当前，xlc/xlC 编译器不提供对静态或自动变量强制进行 16 字节对齐的方法。编译器仅对结构内的 128 位长双精度字段保证相关对齐。OS/400 PASE 版本的 malloc 始终提供 16 字节对齐存储，而且您能够处理堆栈存储的 16 字节对齐。
- 头文件 as400_types.h 也依靠 long long 类型来形成 64 位整数。xlc 编译器选项 -qlonglong 可确保此几何结构（不是运行 xlc 编译器的所有命令的缺省值）。

示例

» 下列示例适用于在 AIX 系统上编译 OS/400 PASE 程序时。如果要使用安装在 OS/400 PASE 中的编译器来编译程序，则不需要为 OS/400 专用的头文件或 OS/400 专用的导出文件的位置指定编译器选项，因为这些文件将在 OS/400 系统上它们的缺省路径 /usr/include/ 和 /usr/lib/ 位置中找到。 «

示例 1: 下列 AIX 系统上的命令创建命名为 *testpgm* 的 OS/400 PASE 程序，它可以使用由 libc.a 导出的 OS/400 专用的接口:

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

本示例假设将 OS/400 专用的头文件复制到 AIX 目录 /usr/include，并且将 OS/400 专用的导出文件复制到 AIX 目录 /mydir。

示例 2: 下列示例假设 OS/400 专用的头文件和导出文件在 /pase/lib 中:

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

示例 3: 下列示例使用相同的选项来构建与示例 2 相同的程序; 但是 `xlc_r` 命令用于多线程程序, 以确保已编译的应用程序与线程安全运行时库相链接:

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

在示例中, 如果要使用 OS/400 PASE 支持 DB2 UDB iSeries 版调用层接口（CLI）, 则还需要在构建命令上指定 `-bI:/pase/include/libdb400.exp`。

`-bI` 伪指令告诉编译器将参数传送给 `ld` 命令。伪指令指定包含从库中导出的且将要通过应用程序导入的符号的导出文件。

将 OS/400 PASE 程序复制到 iSeries 服务器

将想要在 OS/400 PASE 中运行的 AIX 二进制复制到集成文件系统。集成文件系统中提供的所有文件系统在 OS/400 PASE 中均有提供。有关集成文件系统的更多信息, 参见集成文件系统主题。

在跨平台移动文件时, 注意下列会产生问题的差异:

- 如果应用程序对第 12 页的『区分大小写』, 则将它移动到 /QOpenSys 文件系统或已创建为区分大小写的用户定义文件系统中。
- AIX 和 OS/400 在文本文件（例如, 文件和外壳程序脚本）中使用不同的第 13 页的『集成文件系统文件中的行终止字符』。

传送文件

可以使用以下任何方法通过 iSeries 服务器传送和接收 OS/400 PASE 程序和相关文件:

- 文件传送协议（FTP）
- 服务器消息块（SMB） (11参见页面)
- 远程文件系统 (12参见页面)

使用文件传送协议（FTP）复制程序

可以使用 OS/400 FTP 守护程序和客户机来将文件传送至 OS/400 集成文件系统或者从 OS/400 集成文件系统传送文件。以二进制方式传送文件。使用 FTP 子命令 `binary` 来设置此方式。

将文件放入集成文件系统中时, 必须使用命名格式 1（OS/400 FTP 命令的 NAMEFMT 1 子命令）。此格式允许使用 UNIX 路径名, 并将文件传送到流文件。要进入命名格式 1, 可以:

- 更改使用 UNIX 路径名的目录。这会自动将会话置于命名格式 1。使用此方法, 第一个目录以斜杠（/）开始。例如:

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

- 对远程客户机使用 FTP 子命令 `quote site namefmt 1`, 或者将 `namefmt 1` 用作本地客户机。

有关 FTP 的更多信息, 参见 FTP 主题。

使用服务器消息块（SMB）复制程序

OS/400 支持 SMB 客户机和服务器组件。在 NetServer 已配置好且正在运行的情况下，OS/400 PASE 可以通过 /QNTC 文件系统访问网络中的 SMB 服务器。在 UNIX 平台上，要求 SAMBA 服务器提供相同服务。安装已配置和可用的 UNIX 系统（如 AIX），可以使目录和文件用于 OS/400 PASE。

使用远程文件系统复制程序

OS/400 允许将“网络文件系统”（NFS）文件系统安装到集成文件系统文件空间中的安装点。AIX 支持 NFS，如同分布式文件系统（DFS）(TM) 和 Andrew 文件系统（AFS）(R)（使用 DFS 至 NFS 和 AFS 至 NFS 转换程序），以使 OS/400 可以导出和安装这些文件系统。反过来，这让 OS/400 PASE 应用程序使用这些文件系统。通过 OS/400 用户概要文件的用户标识号和组标识号为目录路径或正在访问的文件验证安全性授权。想要确保同一个人通过多个平台访问的用户概要文件在所有系统中具有相同的用户标识。

用作 NFS 服务器，OS/400 是最佳选择。在此种情况下，应从 AIX 系统安装到 OS/400 集成文件系统中的目录上，而且当程序在构建时，AIX 会直接把程序写到 OS/400 上。

注：OS/400 NFS 当前在多线程应用程序中不受支持。

区分大小写

UNIX 系统界面通常区分大写字母和小写字母。在 OS/400 上，情况却并不总是如此。特别是，您应该注意区分大小写可能会导致现有代码复杂化的几种情况。

在目录或文件基础上区分大小写取决于正在 OS/400 上使用的文件系统。/QOpenSys 文件系统是区分大小写的，因此您可以创建区分大小写的用户定义的文件系统（UDFS）。有关各种文件系统的特征的信息，参见集成文件系统主题（特别是，集成文件系统上的文件系统：比较主题）。

还应该注意：OS/400 上的用户标识和组标识总是以大写形式返回。

示例

以下是可能会遇到的因区分大小写而产生的问题的一些示例。

示例 1：在这些示例中，外壳程序将 `readdir()` 所返回的内容与类属名前缀进行字符比较。但是，QSYS.LIB 文件系统返回大写形式的目录项，因此这些项与小写形式的类属名前缀均不匹配。

```
$ 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
```

示例 2：此示例与第一个示例相似，不同之处在于此处是由查找实用程序来进行比较，而不是外壳程序。

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

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

示例 3：`ps` 实用程序预期用户名是区分大小写的，因此不能识别为 `-u` 选项指定的大写名称和由 OS/400 PASE 运行时函数 `getpwuid()` 返回的小写名称之间的匹配关系：

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

集成文件系统文件中的行终止字符

作为 OS/400 PASE 程序的源的 AIX 应用程序要求行（例如，在文件和外壳程序脚本中）以换行（LF）结束。然而，PC 软件和典型的 OS/400 软件经常以回车换行（CRLF）结束行。

与 FTP 一起使用的 CRLF

此差异会在何处导致问题的一个示例是在使用 FTP 来将源文件和外壳程序脚本从 AIX 传送到 iSeries。FTP 标准调用以文本方式发送的数据来在行结束处使用回车换行（CRLF）。在 AIX 上，当 FTP 实用程序处理文本方式的入站文件时，将会除去回车（CR）。OS/400 FTP 始终写明在数据流中提供什么，并始终为文本方式保留 CRLF，它会导致 OS/400 PASE 运行时和实用程序的使用问题。

如果可能，使用二进制方式从 UNIX 系统传送，以避免此问题。在多数情况下，将从个人计算机传送的文本文件中以 CRLF 定界行。第一次将文件传送到 AIX 将更正问题。提供以下变通方法作为从当前目录中的文件中除去 CR 的一种方式:

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

在 iSeries 和 PC 编辑器中使用 CRLF

当使用 iSeries 服务器上的编辑器或工作站（如 Windows(R) 记事本编辑器）上的编辑器编辑文件或外壳程序脚本时，也可能遇到问题。这些编辑器将 CRLF 用作换行分隔符，而不是如 OS/400 PASE 所期望的使用 LF。

多数不将 CRLF 用作换行分隔符的编辑器也可用（例如，ez 编辑器）。参见 IBM PartnerWorld Web 站点上的各种 iSeries tools for developers 列表。

定制 OS/400 PASE 程序来使用 OS/400 功能

如果要使 AIX 应用程序利用系统提供的 OS/400 PASE 共享库所不直接支持的 OS/400 功能，则需要执行一些额外步骤来准备应用程序。

首先，需要定制 AIX 应用程序来调用任何对协调访问 OS/400 专用函数所要求的 OS/400 PASE 运行时函数。

其次，如果正在 AIX 系统上编译 OS/400 PASE 程序，则在编译定制应用程序之前，需要执行以下步骤:

- 『复制头文件』到 AIX 系统
- 第 14 页的『复制导出文件』复制到 AIX 系统

关于如何将 OS/400 PASE 与 OS/400 功能集成的更多信息，参见以下主题:

- 第 20 页的『从 OS/400 PASE 程序调用 OS/400 程序和过程』
- 第 30 页的『OS/400 PASE 程序如何与 OS/400 进行交互』

复制头文件

为唯一支持 OS/400，OS/400 PASE 增大了使用头文件的标准 AIX 运行时。这些由 OS/400 PASE (14参见页面) 和 OS/400 操作系统 (14参见页面)提供。将头文件从 iSeries 服务器上复制到位于头文件搜索路径中的 AIX 机器上。

可以将它们复制到以下 AIX 目录，或编译器的头文件搜索路径中的任何其它目录。

```
/usr/include
```

如果使用的目录不是 /usr/include，则可以使用 AIX 编译器命令的 -I 选项将它添加到头文件搜索路径中。

有关复制文件的更多信息，参见第 11 页的『将 OS/400 PASE 程序复制到 iSeries 服务器』。

复制 OS/400 PASE 头文件

OS/400 PASE 头文件定位在以下 OS/400 目录:

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

OS/400 PASE 提供下列头文件:

as400_protos.h	OS/400 PASE 到 ILE。提供各种 OS/400 专用的函数。
as400_types.h	调用到 ILE 的唯一 OS/400 参数类型 此头文件表明类型 ILEpointer 为 16 字节机器接口 (MI) 指针，它依靠长双精度类型来形成 128 位字段。 在 as400_types.h 中声明的其它类型依靠 8 字节类型来形成 64 位整数。AIX 编译器必须使用选项 -qlngdbl128、-qalign=natural 和 -qlonglong 来运行，以确保在 as400_types.h 中声明的类型的适当大小和对齐。
os400msg.h	发送和接收 OS/400 消息的函数

复制 OS/400 头文件

OS/400 所提供的头文件可在以下目录中找到:

```
/QIBM/include
```

如果应用程序需要任何 OS/400 API 头文件，则必须先将头文件从 EBCDIC 转换到 ASCII，然后再将已转换的文件复制到 AIX 目录。

将 EBCDIC 文本文件转换到 ASCII 的一种方法是使用 OS/400 PASE Rfile 实用程序。

以下示例使用 OS/400 PASE Rfile 实用程序来读取 OS/400 头文件 /QIBM/include/qusec.h，将数据转换为 OS/400 PASE CCSID，除去每行尾部的空格，以及将结果写入字节流文件 ascii_qusec.h:

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

复制导出文件

将导出文件从 iSeries 服务器复制到 AIX 目录。

位于下列 OS/400 目录中的导出文件是建议用来构建要求访问特定于 OS/400 函数的应用程序的方法:

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

可以将这些文件复制到任何 AIX 目录。在 AIX ld 命令（或编译器命令）上使用 -bI: 选项来定义在 AIX 系统上的共享库中找不到的符号。

OS/400 PASE 提供以下导出文件。

as400_libc.exp	libc.a 中的 OS/400 专用的函数的导出文件 as400_libc.exp 文件定义所有从 OS/400 PASE 版本的 libc.a 中的导出，而这些导出不是那些 AIX 版本的库导出的。

libdb400.exp	OS/400 数据库函数的导出文件 libdb400.exp 文件定义从 OS/400 PASE libdb400.a 库（DB2 UDB iSeries 版调用层接口（CLI）支持）的导出。

有关复制文件的更多信息，参见第 11 页的『将 OS/400 PASE 程序复制到 iSeries 服务器』。

访问 OS/400 函数的 OS/400 PASE API

OS/400 PASE 提供许多用于访问 ILE 代码和其它 OS/400 函数的 API。使用哪些 API 取决于您自己想做多少准备工作和结构构建工作（相对于您想要编译器为您做多少准备工作和结构构建工作）。有关详细信息，参见 OS/400 PASE API。

在 OS/400 环境中使用 OS/400 PASE 程序

您的 OS/400 PASE 程序可以调用在作业中运行的其它 OS/400 程序，而其它 OS/400 程序可以调用您的 OS/400 PASE 程序中的过程。有关如何将 OS/400 PASE 程序集成到计算环境中的信息，参见以下主题:

『运行 OS/400 PASE 程序和过程』
提供有关在作业中启动 OS/400 PASE 程序以及从 ILE 程序中调用 OS/PASE 过程的信息和示例。

第 20 页的『从 OS/400 PASE 程序调用 OS/400 程序和过程』
提供有关从 OS/400 PASE 程序中调用 ILE 过程、OS/400 程序和 CL 命令的信息和示例。

第 30 页的『OS/400 PASE 程序如何与 OS/400 进行交互』
提供有关 OS/400 PASE 程序如何使用 OS/400 功能及与其交互的信息。

运行 OS/400 PASE 程序和过程

OS/400 PASE 提供了多种方法用于运行 OS/400 PASE 程序。

第 16 页的『使用 QP2SHELL() 运行 OS/400 PASE 程序』
在调用 OS/400 程序的作业中运行 OS/400 PASE 程序的 OS/400 程序。

第 17 页的『使用 QP2TERM() 运行 OS/400 PASE 程序』
在交互外壳程序环境中运行 OS/400 PASE 程序的 OS/400 程序。

第 17 页的『从 OS/400 程序内运行 OS/400 PASE 程序』
从 ILE 过程内部调用以启动和运行 OS/400 PASE 程序的 ILE 过程。

第 18 页的『从 OS/400 程序内调用 OS/400 PASE 过程』
从 ILE 过程内部调用，在已运行 OS/400 PASE 环境的作业中运行 OS/400 PASE 程序的 ILE 过程。

第 19 页的『使用环境变量』说明 OS/400 PASE 环境如何同 OS/400 环境相互影响。

允许您处理 OS/400 PASE 程序的 ILE 过程

OS/400 PASE 提供了多个 ILE 过程 API，允许您的 ILE 代码访问 OS/400 PASE 服务（无须在 OS/400 PASE 程序中特殊编程）:

- Qp2ptrsize
- Qp2jobCCSID
- Qp2paseCCSID

- Qp2errnop
- Qp2malloc
- Qp2free
- Qp2dlopen
- Qp2dlsym
- Qp2dlclose
- Qp2dlerror

有关更多信息，参见 OS/400 PASE ILE 过程 API。

附加至 ILE 线程

在 OS/400 PASE 程序中，您可以从在不是由 OS/400 PASE 创建的线程（例如，Java 线程或 ILE pthread_create 创建的线程）中运行的 ILE 代码中调用过程。Qp2CallPase 会自动将 ILE 线程附加至 OS/400 PASE（创建相应的 OS/400 PASE pthread 结构），但只有在启动 OS/400 PASE 程序时将 OS/400 PASE 环境变量 PASE_THREAD_ATTACH 设置为 Y 时才会这样。

将结果从 OS/400 PASE 返回至 OS/400 程序

使用 OS/400 _RETURN() 函数，您可以调用 OS/400 PASE 程序并返回结果，而无须结束 OS/400 PASE 环境。这样，您可以启动 OS/400 PASE 程序，然后在 QP2SHELL2（而不是 QP2SHELL）或 Qp2RunPase API 返回后在该程序中调用过程（使用 Qp2CallPase）。 

使用 QP2SHELL() 运行 OS/400 PASE 程序

使用“运行 OS/400 PASE 外壳程序”（QP2SHELL 或  QP2SHELL2  ）程序来从任何 OS/400 命令行和任何高级语言程序、批处理作业或交互式作业内运行 OS/400 PASE 程序。这些程序在调用 OS/400 PASE 程序的作业中运行该 OS/400 PASE 程序。OS/400 PASE 程序的名称作为参数在程序上传送。关于如何使用此程序的详细信息，参见 QP2SHELL() 和 QP2SHELL2() 描述。

QP2SHELL() 程序在新激活组中运行 OS/400 PASE 程序。  QP2SHELL2() 程序在调用者的激活组中运行。 

以下示例从 OS/400 命令行运行 ls 命令:

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

使用变量将值传送给 QP2SHELL()

如果使用 CL 变量将值传送给 QP2SHELL()，则变量必须是以空终止。例如，需要以下列方式来编写以上示例的代码:

```
PGM

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

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

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

ENDIT:
ENDPGM
```

使用 QP2TERM() 运行 OS/400 PASE 程序

使用 QP2TERM() 程序启动 OS/400 PASE 交互式终端会话。以下命令将缺省 Korn 外壳程序提示（/QOpenSys/usr/bin/sh）写入屏幕:

```
call qp2term
```

从本提示符，在单独的批处理作业中运行 OS/400 PASE 程序。QP2TERM() 使用交互式作业来显示批处理作业中的文件 stdin、stdout 和 stderr 的输出以及接受其输入。

Korn 外壳程序是缺省值，您也可以随意指定任何要运行的 OS/400 PASE 程序的路径名，以及任何传送给程序的自变量字符串。

可以从使用 QP2TERM() 启动的交互式会话中运行任何 OS/400 PASE 程序和任何实用程序；将 stdout 和 stderr 写入终端并在终端屏幕中滚动。

从 OS/400 程序内运行 OS/400 PASE 程序

使用 Qp2RunPase() API 来运行 OS/400 PASE 程序。指定程序名、自变量字符串和环境变量。关于如何在 ILE 程序中使用它的详细信息，参见 Qp2RunPase() API 描述。

Qp2RunPase() API 在调用它的作业中运行 OS/400 PASE 程序。它装入 OS/400 PASE 程序（包括任何必要的共享库），然后将控件传送给程序。

此 API 对 OS/400 PASE 如何运行提供了比 QP2SHELL() and QP2TERM() 更多的控制。

如何能够在 ILE 程序中使用此 API 的示例，参见『示例: OS/400 程序内运行 OS/400 PASE 程序』。

示例: OS/400 程序内运行 OS/400 PASE 程序: 以下 ILE 程序（参见第 44 页的『示例』）调用 OS/400 PASE 程序。以下示例是此程序调用的 OS/400 PASE 代码示例。

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

/* include file for QP2RunPase(). */

#include <qp2user.h>

/*****************************************
  Sample:
  A simple ILE C program to invoke an OS/400
  PASE program using QP2RunPase() and
  passing one string parameter.
  Example compilation:
    CRTCMOD MODULE(MYLIB/SAMPLEILE) SRCFILE(MYLIB/QCSRC)
    CRTPGM PGM(MYLIB/SAMPLEILE)
*****************************************/

void main(int argc, char*argv[])
 {
  /* Path name of PASE program */
  char *PasePath = "/home/samplePASE";
```

```
  /* Return code from QP2RunPase() */
  int rc;
  /* The parameter to be passed to the
     OS/400 PASE program */
  char *PASE_parm = "My Parm";
  /* Argument list for OS/400 PASE program,
     which is a pointer to a list of pointers */
  char **arg_list;
  /* allocate the argument list */
  arg_list =(char**)malloc(3 * sizeof(*arg_list));
  /* set program name as first element. This is a UNIX convention */
  arg_list[0] = PasePath;
  /* set parameter as first element */
  arg_list[1] = PASE_parm;
  /* last element of argument list must always be null */
  arg_list[2] = 0;
  /* Call OS/400 PASE program. */
   rc = Qp2RunPase(PasePath, /* Path name */
      NULL,           /* Symbol for calling to ILE, not used in this sample */
      NULL,           /* Symbol data for ILE call, not used here */
      0,              /* Symbol data length for ILE call, not used here */
      819,            /* ASCII CCSID for OS/400 PASE */
      arg_list,       /* Arguments for OS/400 PASE program */
      NULL);          /* Environment variable list, not used in this sample */
 }
```

以下 OS/400 PASE 程序（参见第 44 页的『示例』）被上面的 ILE 程序所调用。

```
#include <stdio.h>

/*****************************************
  Sample:
  A simple OS/400 PASE Program called from
  ILE using QP2RunPase() and accepting
  one string parameter.
  The ILE sample program expects this to be
  located at /home/samplePASE. Compile on
  AIX, then ftp to OS/400.
  To ftp use the commands:
  > binary
  > site namefmt 1
  > put samplePASE /home/samplePASE
*****************************************/

int main(int argc, char *argv[])
{
    /* Print out a greeting and the parameter passed in. Note argv[0] is the program
       name, so, argv[1] is the parameter */
     printf("Hello from OS/400 PASE program %s. Parameter value is \"%s\".\n", argv[0], argv[1]);

     return 0;
}
```

从 OS/400 程序内调用 OS/400 PASE 过程

Qp2RunPase() API 最初在作业中启动和运行 OS/400 PASE 程序。如果在此作业中 OS/400 PASE 已经是活动的，则它将返回一个错误。

要在已经运行 OS/400 PASE 程序的作业中调用 OS/400 PASE 过程，需使用 Qp2CallPase() 和 Qp2CallPase2() API。

如何使用 Qp2CallPase() API 的示例，参见第 19 页的『示例: 从 OS/400 程序内调用 OS/400 PASE 过程』。

示例: 从 OS/400 程序内调用 OS/400 PASE 过程: 以下 ILE 程序（参见第 44 页的『示例』）调用 OS/400 PASE 过程。

```
#include <stdio.h>
#include <qp2shell2.h>
#include <qp2user.h>
#define JOB_CCSID 0

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

    /*
     * Call QP2SHELL2 to run the OS/400 PASE program
     * /usr/lib/start32, which starts OS/400 PASE in
     * 32-bit mode (and leaves it active on return)
     */
    QP2SHELL2("/usr/lib/start32");

    /*
     * Qp2dlopen opens the global name space (rather than
     * loading a new shared executable) when the first
     * argument is a null pointer.  Qp2dlsym locates the
     * function descriptor for the OS/400 PASE getpid
     * subroutine (exported by shared library libc.a)
     */
    id = Qp2dlopen(NULL, QP2_RTLD_NOW, JOB_CCSID);
    getpid_pase = Qp2dlsym(id, "getpid", JOB_CCSID, NULL);

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

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

使用环境变量

OS/400 PASE 环境变量独立于 ILE 环境变量。在一个环境中设置的变量对另一个环境无效。然而，可以将变量从 ILE 中复制到 OS/400 PASE 中，这取决于第 15 页的『运行 OS/400 PASE 程序和过程』所使用的方法。

交互式 OS/400 PASE 会话中的环境变量

仅当 ILE 环境变量用 QP2SHELL() 和 QP2TERM() 启动时，它可被传送给 OS/400 PASE。在启动 OS/400 PASE 之前，使用“使用环境变量”（WRKENVVAR） 命令来根据需要更改、添加或删除环境变量。

被调用的 OS/400 PASE 会话中的环境变量

当 OS/400 PASE 从程序调用（使用 Qp2RunPase() API）中启动时，已经完成了对环境变量的控制。可以传送与从中调用 OS/400 PASE 程序的 ILE 环境无关的环境变量。

在运行 CL 命令之前将环境变量复制到 ILE

在使用 systemCL 运行时函数运行 CL 命令之前，可以将 OS/400 PASE 环境变量复制到 ILE 环境中。这也是 OS/400 PASE `system` 实用程序的缺省行为。

附加信息，参见 OS/400 PASE 环境变量主题。

从 OS/400 PASE 程序调用 OS/400 程序和过程

OS/400 PASE 提供用于调用 ILE 过程、Java 程序、OPM 程序、OS/400 API 和 CL 命令（它们使您可以全方位访问 OS/400 函数）的方法。

以下主题提供从 OS/400 PASE 环境进行调用的指令和示例。

『调用 ILE 过程』
从 OS/400 PASE 调用 ILE 过程之前，需要确保设置 ILE 过程，以便处理通过 OS/400 PASE 程序进行的调用。还需要在已编译的 AIX 程序中设置程序变量和结构。

第 28 页的『从 OS/400 PASE 调用 OS/400 程序』
可以从 OS/400 PASE 程序内调用 OS/400 程序。

第 29 页的『从 OS/400 PASE 运行 OS/400 命令』
可以从 OS/400 PASE 程序内运行 CL 命令。

OS/400 程序和过程的一般配置需求

由于以下原因，当从 OS/400 PASE 程序环境到 OS/400 环境进行调用时，通常应该确保使用 *CALLER 来为激活组编译 OS/400 程序:

- 只有在启动了 OS/400 PASE（由 Qp2RunPase API 调用）的激活组中运行的代码可以使用 ILE API（如 Qp2CallPase）与 OS/400 PASE 程序进行交互。
- 如果 ILE 运行时需要在多线程作业（以及由 OS/400 PASE `fork` 创建的所有作业都是支持多线程的）中消除激活组，则它可能会结束整个作业（同时也结束 OS/400 PASE）。通过使用 ACTGRP(*CALLER)，可以防止作业在您想要将其结束之前就结束了。

通过使用 `systemCL` 运行时函数来运行不支持多线程的单个作业中的 CL 命令（包括 CALL 命令），您可以避免与在支持多线程的作业中运行有关的问题。

调用 ILE 过程

要从 OS/400 PASE 程序调用 ILE 过程，在代码中做出以下 API 调用:

1. 将绑定程序装入与启动 OS/400 PASE 的过程相关联的 ILE 激活组。使用 _ILELOAD() API 来完成此操作。如果绑定程序在启动了 OS/400 PASE 的激活组中已经是活动的，则可以不必执行此步骤。在此情况下，可以继续执行 _ILESYM 步骤（使用激活标志参数的零值来搜索当前激活组中所有活动的绑定程序中的全部符号）。
2. 在 ILE 绑定程序处于激活的情况下查找导出的符号，并为符号返回指向数据或过程的 16 字节标记指针。使用 _ILESYM() API 来完成此操作。

3. 调用 ILE 过程将控制从 OS/400 PASE 程序传送到 ILE 过程。使用 _ILECALL() 或 _ILECALLX() API 来完成此操作。

此外，在从 OS/400 PASE 程序调用 ILE 过程时，必须执行下列任务:

- 为太字节空间启用 ILE 过程
- 将文本转换为相应的 CCSID
- 设置变量和结构

为太字节空间启用 ILE 过程

从 OS/400 PASE 调用的所有 ILE 模块必须在太字节空间选项设置为 *YES 的情况下进行编译。如果没有以此方式编译 ILE 模块，则将会在 OS/400 PASE 应用程序的作业记录中接收到 MCH4433 错误消息（目标程序 &2 的无效存储型号）。有关更多信息，参见 ILE Concepts 一书。

将文本转换为相应的 CCSID

在 ILE 和 OS/400 PASE 之间传送的文本在传送之前可能需要转换为相应的 CCSID。未进行此类转换将会导致字符变量包含不可破译的值。

设置变量和结构

要从 OS/400 PASE 程序调用 ILE，需要设置变量和结构。必须确保将所需的头文件复制到 AIX 系统，并且必须设置特征符、结果类型和自变量列表变量。

- **头文件**: OS/400 PASE 程序应该包括第 13 页的『复制头文件』，以便调用 ILE。as400_type.h 头文件包含用于 OS/400 专用的接口类型定义。
- **特征符**: 特征符结构包含在 OS/400 PASE 和 ILE 之间传送的自变量的顺序和类型的描述。由正在调用的 ILE 过程控制的类型的编码可在 as400_types.h 头文件中找到。如果特征符包含短于 4 个字节的定点自变量或短于 8 个字节的浮点自变量，则 ILE C 代码需要使用以下编译指示进行编译:

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

  在没有此编译指示的情况下，ILE 的标准 C 连接要求将 1 字节和 2 字节整数自变量拓展到 4 个字节，将 4 字节的浮点自变量拓展到 8 个字节。
- **结果类型**: 结果类型简单明了，与 C 中的返回类型的工作方式非常相似。
- **自变量列表**: 自变量列表必须是具有正确顺序的字段的结构，且带有特征符数组中的项所指定的类型。

  可以使用 size_ILEarglist() 和 build_ILEarglist() API 来动态地构建基于特征符的自变量列表。

有关举例说明从 OS/400 PASE 调用 ILE 过程的进程的示例，参见『示例: 调用 ILE 过程』。

示例: 调用 ILE 过程: 下列代码示例（参见第 44 页的『示例』）显示调用作为服务程序一部分的 ILE 过程 (25参见页面)的 OS/400 PASE 代码 (21参见页面)以及创建程序的编译器命令 (27参见页面)。在示例中，有两个 UNIX 过程。每个过程演示了使用 ILE 过程的不同方式，但两个过程调用的是相同的 ILE 过程。第一个过程演示了使用 OS/400 PASE 提供的方法为 _ILECALL API 构建数据结构。第二个过程手工构建自变量列表。

OS/400 PASE C 代码

下列示例代码中散布的是解释代码的注释。确保在输入或复查示例时阅读这些注释。

```
/* Name: PASEtoILE.c
 *
 * You must use compiler options -qalign=natural and -qldbl128
 * to force relative 16-byte alignment of type long double
```

```
 * (used inside type ILEpointer)
 */

#include <stdlib.h>
#include <malloc.h>
#include <sys/types.h>
#include <stdio.h>
#include "as400_types.h"
#include "as400_protos.h"

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

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

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

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

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

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

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

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

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

   /*
    * Save the current PID in static storage so we
```

```
    * can determine when to re-initialize (after fork)
    */
   init_pid = getpid();
}

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

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

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

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

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

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

/*
 * ILEarglistSt defines the structure of the ILE argument list.
 * xlc provides 16-byte (relative) alignment of ILEpointer
 * member fields because ILEpointer contains a 128-bit long
 * double member. Explicit pad fields are only needed in
 * front of structure and union types that do not naturally
 * fall on ILE-mandated boundaries
 */
typedef struct {
    ILEarglist_base base;
    int32 arg1;
    /* implicit 12-byte pad provided by compiler */
    ILEpointer arg2;
    float64 arg3;
    uint8 arg4;
    char filler[7]; /* pad to 8-byte alignment */
    aggregate arg5; /* 5-byte aggregate (8-byte align) */
    /* implicit 1-byte pad provided by compiler */
    int16 arg6;
} ILEarglistSt;

/*
 * wrapper_2 accepts the same arguments and returns
 * the same result as the ILE procedure it calls. This
 * method uses a customized or declared structure for the
 * ILE argument list to improve execution efficiency and
 * avoid heap storage leaks if an exception or signal occurs
 */
int wrapper_2(int arg1, void *arg2, double arg3,
                        char arg4, aggregate arg5, short arg6)
```

```
{
    /*
     * xlc does not guarantee 16-byte alignment for
     * automatic (stack) variables of any type, so we
     * find an aligned area in an oversized buffer
     */
    char ILEarglist_buf[sizeof(ILEarglistSt) + 15];
    ILEarglistSt *ILEarglist = (ILEarglistSt*)ROUND_QUAD(ILEarglist_buf);
    /*
     * Assignment statements are faster than calling
     * build_ILEarglist()
     */
    ILEarglist->arg1 = arg1;
    ILEarglist->arg2.s.addr = (address64_t)arg2;
    ILEarglist->arg3 = arg3;
    ILEarglist->arg4 = arg4;
    ILEarglist->arg5 = arg5;
    ILEarglist->arg6 = arg6;
    /*
     * Use a saved PID value to check if the ILE pointer
     * is set. ILE procedure pointers inherited by the
     * child process of a fork() are not usable because
     * they point to an ILE activation group in the parent
     * process
     */
    if (getpid() != init_pid)
    do_init();
    /*
     * _ILECALL calls the ILE procedure. The stack may
     * be unwound, but no heap storage is orphaned if
     * an exception or signal occurs
     */
    _ILECALL(ILEtarget,
        &ILEarglist->base,
        signature,
        result_type);
    if (ILEarglist->base.result.s_int32.r_int32 == 1)
      printf("The results of best_wrapper function is: %s\n", arg2);
    else if ( ILEarglist->base.result.s_int32.r_int32 == 0)
    printf("ILE received other than 1 or 2 for version.\n");
    else printf("The db file never opened.\n");
    return ILEarglist->base.result.s_int32.r_int32;
    }
    void main () {
      int version,
                   result2;
      char dbText[ 25 ];
      double dblNumber = 5.999;
      char justChar = 'a';
      short shrtNumber = 3;
      aggregate agg;
      strcpy( dbText, "none" );

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

ILE C 代码

现在在 OS/400 系统上为此示例写 ILE C 代码。需要在其中写代码的库中的源物理文件。同样，注释会散布在 ILE 示例中。这些注释是理解代码的关键。您应该在输入或复查源时复查它们。

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

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

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

/*
 * The arguments and function result for this ILE procedure
 * must be equivalent to the values presented to _ILECALL
 * function in the OS/400 PASE program
 */
int ileProcedure(int         arg1,
                 char        *arg2,
                 double      arg3,
                 char        arg4[2],
                 aggregate   arg5,
                 short       arg6)
{
    char         fromcode[33];
    char         tocode[33];
    iconv_t      cd;     /* conversion descriptor */
    char         *src;
    char         *tgt;
    size_t       srcLen;
    size_t       tgtLen;
    int          result;

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

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

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

创建程序的编译器命令

在编译 OS/400 PASE 程序时，必须使用编译器选项 `-qalign=natural` 和 `-qldbl128` 来对 ILEpointer 类型中使用的长双精度类型强制进行相对的 16 字节对齐。OS/400 中的 ILE 要求这种对齐。对于选项 `-bI:`，应该输入已在其中保存 as400_libc.exp 的路径名:

```
xlc -o PASEtoILE -qldbl128 -qalign=natural
       -bI:/afs/rich.xyz.com/usr1/shupe/PASE/as400_libc.exp
       PASEtoILE.c
```

编译 ILE C 模块和服务程序时，使用太字节空间选项来编译它们。否则，OS/400 PASE 不能与它们进行交互。

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

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

最后，必须编译 DDS 并至少传播一条数据记录:

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

从 OS/400 PASE 调用 OS/400 程序

在创建 OS/400 PASE 应用程序时，可以利用现有的 OS/400 程序（*PGM 对象）：

- 使用 systemCL 函数来运行 CL CALL 命令。有关信息和示例，参见第 29 页的『从 OS/400 PASE 运行 OS/400 命令』。
- 使用 _PGMCALL 运行时函数从 OS/400 PASE 程序内调用 OS/400 程序。此方法提供比 systemCL 运行时函数更快的执行速度，但是它不执行字符串自变量的自动转换，而且不提供在不同作业中调用程序的功能。

有关如何在 OS/400 PASE 程序中使用 _PGMCALL 运行时函数来调用命令的示例，参见『示例: 从 OS/400 PASE 调用 OS/400 程序』。 

示例: 从 OS/400 PASE 调用 OS/400 程序: 下列示例（参见第 44 页的『示例』）显示如何在 OS/400 PASE 程序中使用 _PGMCALL 运行时函数调用程序:

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

   The QSZRTVPR API is used to retrieve information about OS/400
   software product loads.  Refer to the QSZRTVPR API documentation for
   specific information regarding the input and output parameters needed
   to call the API */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "as400_types.h"
#include "as400_protos.h"

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

   /* OS/400 API's (including QSZRTVPR) typically expect character
      parameters to be in EBCDIC.  However, character constants in
      OS/400 PASE programs are typically in ASCII.  So, declare some
      CCSID 37 (EBCDIC) character parameter constants that will be
      needed to call QSZRTVPR */

   /* format[] is input parameter 3 to QSZRTVPR and is
      initialized to the text 'PRDR0100' in EBCDIC */
   const char format[] =
      {0xd7, 0xd9, 0xc4, 0xd9, 0xf0, 0xf1, 0xf0, 0xf0};

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

      This value indicates we want to check the code load for Option 33
      of the currently installed OS/400 release */
   const char prodinfo[] =
      {0x5c, 0xd6, 0xd7, 0xe2, 0xe8, 0xe2, 0x40, 0x5c, 0xc3,
       0xe4, 0xd9, 0x40, 0x40, 0xf0, 0xf0, 0xf3, 0xf3, 0x5c,
       0xc3, 0xd6, 0xc4, 0xc5, 0x40, 0x40, 0x40, 0x40, 0x40};

   /* installed will be compared with the "Load State" field of the
      information returned by QSZRTVPR and is initialized to the text
      '90' in EBCDIC */
   const char installed[] = {0xf9, 0xf0};

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

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

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


   /* qszrtvpr_pointer will contain the OS/400 16-byte tagged system
      pointer to QSZRTVPR */
   ILEpointer qszrtvpr_pointer;

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

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

   /* Set the OS/400 pointer to the QSYS/QSZRTVPR *PGM object */
   rc = _RSLOBJ2(&qszrtvpr_pointer,
                 RSLOBJ_TS_PGM,
                 "QSZRTVPR",
                 "QSYS");

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

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

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

   /* Call the OS/400 QSZRTVPR API from OS/400 PASE */
   rc = _PGMCALL(&qszrtvpr_pointer,
                 (void*)&qszrtvpr_argv,
                 0);

   /* Check the contents of bytes 63-64 of the returned information.
      If they are not '90' (in EBCDIC), the code load is NOT correctly
      installed */
   if (memcmp(&rcvr[63], &installed, 2) != 0)
      printf("OS/400 Option 33 is NOT installed\n");
   else
      printf("OS/400 Option 33 IS installed\n");

   return(0);
}
```

«

从 OS/400 PASE 运行 OS/400 命令

通过运行使用 OS/400 函数的控制语言（CL）命令，可以扩展 OS/400 PASE 程序的功能。使用 systemCL 运行时函数运行来自 OS/400 PASE 程序内的 OS/400 命令。

» 从 OS/400 PASE 运行 OS/400 命令时，systemCL 运行时函数会自动处理字符串自变量从 ASCII 至 EBCDIC 的转换，并允许在不同的作业中调用该程序。 «

有关如何在 OS/400 PASE 程序中运行 CL 命令的示例，参见『示例: 从 OS/400 PASE 运行 OS/400 命令』。

示例: 从 OS/400 PASE 运行 OS/400 命令: 下列示例（参见第 44 页的『示例』）显示如何在 OS/400 PASE 程序中调用命令:

```
/* sampleCL.c

   example to demonstrate use of sampleCL to run a CL command

   Compile with a command similar to the following.
   xlc -o sampleCL -I /whatever/pase -bI:/whatever/pase/as400_libc.exp sampleCL.c

   Example program using QP2SHELL() follows.
   call qp2shell ('sampleCL' 'wrkactjob')
*/
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <as400_types.h>  /* PASE header */
#include <as400_protos.h> /* PASE header */

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

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

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

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

OS/400 PASE 程序如何与 OS/400 进行交互

定制 OS/400 PASE 程序以使用 OS/400 功能时，需要考虑程序将与它们进行交互的方式。以下主题提供基本指南，并提供到 “iSeries 信息中心” 中的 OS/400 详细系统信息的链接:

- 通信
- 数据库
- 数据编码
- 文件系统
- 全球化
- 消息服务
- 打印
- 伪终端（PTY）
- 安全性

- 工作管理

通信

» OS/400 PASE 支持与 AIX 相同的语法以进行套接字通信。这可能不会与其它 UNIX(TM) 系统在每个细节上都匹配。

OS/400 PASE 套接字支持可以与 AIX 的套接字实现相媲美，但 OS/400 PASE 使用 OS/400 的套接字实现（而非 AIX 内核套接字实现），而且这会造成与 AIX 行为的一些细微差别。

OS/400 的套接字实现支持 UNIX 98 和 Berkeley 软件分发（BSD）套接字。在大多数情况下，OS/400 PASE 通过采用 AIX 实现的行为来解决这些风格上的差别。

此外，用于运行应用程序的用户概要文件必须具有 *IOSYSCFG 特权，以便将层参数指定为 IPPROTO_IP，并且将 option_value 参数指定为套接字 API 上的 IP_OPTIONS。有关 OS/400 上套接字的使用的详细信息，参见套接字编程主题。特别是要参见 Berkeley 软件分发（BSD）兼容性和 UNIX 98 兼容性主题。 «

数据库

OS/400 PASE 支持 iSeries 调用层接口（CLI）的 DB2 UDB。AIX 和 OS/400 上的 DB2 CLI 不是相互适合的子集，因此在几个接口中存在细微差别，而且一些在一个实现中的 API 可能在另一个实现中不存在。因此，应该考虑以下几点:

- 可以生成代码，但不能在 AIX 自身测试代码。而是必须通过 OS/400 PASE 内的平台测试代码。
- » 必须使用头文件 `sqlcli.h` 的 OS/400 版本编译。使用该头文件的 AIX 版本所编译的程序将不能在 OS/400 PASE 上运行。 «

缺省情况下，OS/400 是一个 EBCDIC 编码系统，而 AIX 是基于 ASCII 的。这个差异导致经常需要在 OS/400 数据库（DB2 UDB iSeries 版）和 OS/400 PASE 应用程序之间进行数据转换。

在 DB2 CLI 的 OS/400 PASE 实现过程中，OS/400 PASE 所提供的库例程自动执行从 ASCII 到 EBCDIC 的数据转换，并备份字符数据。转换是基于正被访问的数据的已标记的 CCSID 和 OS/400 PASE 程序在其下运行的 ASCII CCSID。如果数据库被标记或标记为 CCSID 65535，则不会发生自动转换。应用程序要做的是了解数据的编码格式以及做任何必要的转换。

使用 CCSID

当使用 Qp2RunPase() API 时，必须显式地指定 OS/400 PASE CCSID。

» 调用 API 程序 QP2TERM、QP2SHELL 或 QP2SHELL2 之前，可以通过设置 ILE 环境中的这两个变量来控制 OS/400 PASE CCSID:

- PASE_LANG
- QIBM_PASE_CCSID

如果 ILE 环境省略了这两个变量中的一个或全部，则缺省情况下，QP2TERM、QP2SHELL 和 QP2SHELL2 将使用作业语言和作业的 CCSID 属性的最佳 OS/400 PASE 等价值来设置 OS/400 PASE CCSID 和 OS/400 PASE 环境变量 `LANG`。 «

更多信息，参见 QP2TERM() 和 QP2SHELL() 程序描述。

» 使用 _SETCCSID() 函数，ibc.a 的扩展使得 OS/400 PASE 应用程序有能力更改应用程序正在运行的 CCSID,

另一个扩展使 OS/400 PASE 应用程序无须更改应用程序的 CCSID 就能够重设 DB2 CLI 内部转换。SQLOverrideCCSID400() 函数以单个参数的形式接受重设 CCSID 的整数。 «

注意: CCSID 重设函数 SQLOverrideCCSID400() 必须在任何其它为其所设的 SQLx() API 生效之前被调用；否则，请求将被忽略。

在 OS/400 PASE 程序中使用 DB2 UDB iSeries 版 CLI

要在 OS/400 PASE 程序中使用 DB2 CLI，需要在编译源代码之前将第 13 页的『复制头文件』 和第 14 页的『复制导出文件』 复制到 AIX 系统中。DB2 CLI 库例程在 OS/400 PASE 环境的 libdb400.a 中，并且通过使用 pthread 接口（提供线程安全）来实现。 »大多数 OS/400 PASE CLI 函数调用相应的 ILE CLI 函数来执行所需要的操作。 «

有关 DB2 UDB 调用层接口的更多信息，参见 DB2 UDB iSeries 版 SQL 调用层接口（ODBC）主题。

欲获得 OS/400 PASE 是如何使用 DB2 UDB iSeries 版 SQL 调用层接口访问 DB2 UDB iSeries 版的示例，参见『示例: 在 OS/400 PASE 程序中调用 DB2 UDB iSeries 版 CLI 函数』。

示例: 在 OS/400 PASE 程序中调用 DB2 UDB iSeries 版 CLI 函数: 以下示例（参见第 44 页的『示例』）显示使用 DB2 UDB iSeries 版 SQL 调用层接口访问 DB2 UDB iSeries 版 OS/400 PASE 程序。

```
/* OS/400 PASE DB2 UDB for iSeries example program
 *
 * To show an example of an OS/400 PASE program that accesses
 * OS/400 DB2 UDB via SQL CLI
 *
 * Program accesses iSeries Access data base, QIWS/QCUSTCDT, that
 * should exist on all systems
 *
 * Change system name, userid, and password in fun_Connect()
 * procedure to valid parms
 *
 * Compilation invocation:
 *
 * xlc -I./include -bI:./include/libdb400.exp -o paseclidb4 paseclidb4.c
 *
 * FTP in binary, run from QP2TERM() terminal shell
 *
 * Output should show all rows with a STATE column match of MN
 */

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

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

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

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

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

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

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

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

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

SQLRETURN fun_Connect()
{
        static char *pszId = "fun_Connect()";
        SQLCHAR chs_As400System[ SQL_MAX_DSN_LENGTH ];
        SQLCHAR chs_UserName[ SQL_MAX_UID_LENGTH ];
        SQLCHAR chs_UserPassword[ SQL_MAX_PWD_LENGTH ];
        nml_ReturnCode = SQLAllocEnv( &nml_HandleToEnvironment );
        if ( nml_ReturnCode != SQL_SUCCESS ) {
           printf( "%s: SQLAllocEnv() succeeded\n", pszId );
           fun_PrintError( SQL_NULL_HSTMT );
           printf( "%s: Terminating\n", pszId );
           return SQL_ERROR;
      } else {
           printf( "%s: SQLAllocEnv() succeeded\n", pszId );
      } /* endif */

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

      nml_ReturnCode = SQLAllocConnect( nml_HandleToEnvironment,
                                        &nml_HandleToDatabaseConnection );

      if ( nml_ReturnCode != SQL_SUCCESS ) {
         printf( "%s: SQLAllocConnect\n", pszId );
         fun_PrintError( SQL_NULL_HSTMT );
         nml_ReturnCode = fun_ReleaseEnvHandle();
```

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

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

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

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

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

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

    Nmi_vParam = SQL_TRUE;
    nml_ReturnCode = SQLSetStmtOption( nml_HandleToSqlStatement,
                                       SQL_ATTR_CURSOR_SCROLLABLE,
                                       ( SQLINTEGER * ) &Nmi_vParam );
    if ( nml_ReturnCode != SQL_SUCCESS ) {
       printf( "%s: SQLSetStmtOption() failed\n", pszId );
```

```
    fun_PrintError( nml_HandleToSqlStatement );
    nml_ReturnCode = fun_ReleaseStmHandle();
    printf( "%s: Terminating\n", pszId );
    return SQL_ERROR;
} else {
    printf( "%s: SQLSetStmtOption() succeeded\n", pszId );
} /* endif */

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

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

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

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

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

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

    return SQL_SUCCESS;
} /* end fun_Process */

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

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

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

     return nml_ReturnCode;
} /* end fun_DisConnect */

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

     nml_ReturnCode = SQLFreeEnv( nml_HandleToEnvironment );
     if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLFreeEnv() failed\n", pszId );
        fun_PrintError( SQL_NULL_HSTMT );
        return SQL_ERROR;
   } else {
```

```
        printf( "%s: SQLFreeEnv() succeeded\n", pszId );
        return SQL_SUCCESS;
   } /* endif */
} /* end fun_ReleaseEnvHandle */

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

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

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

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

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

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

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

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

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

数据编码

大多数 UNIX 系统使用 ASCII 字符编码。 大多数 OS/400 函数使用 EBCDIC 字符编码。可以为一些 OS/400 对象类型指定 CCSID（编码字符集识别码）值来为对象中的字符数据指定特定的编码。

OS/400 PASE 字节流文件具有大多数 OS/400 PASE 外的系统接口使用的 CCSID 属性，根据需要转换从文件读取的文本数据或写入文件的文本数据。OS/400 PASE 不能为从流文件读取的数据或写入流文件的数据（与 AIX 一致）执行 CCSID 转换，但它可以将由 OS/400 PASE 程序所创建的任何字节流文件的 CCSID 属性设置为当前 OS/400 PASE CCSID 值，以使其它系统函数正确处理文件中的 ASCII 文本。

如果使用随 OS/400 PASE 共享库交付的 AIX API，则 OS/400 PASE 可处理大部分数据转换。 对于任何不能由 OS/400 PASE 运行时自动处理的字符数据转换，OS/400 PASE 程序可以使用在共享库 `libiconv.a` 中提供的 `iconv` 函数。例如，在调用 OS/400 API 函数（使用 _ILECALLX 或 _PGMCALL）之前，OS/400 PASE 应用程序通常需要将字符串转换到 EBCDIC。

文件系统

OS/400 PASE 程序可以访问任何可通过集成文件系统访问的文件或资源，包括 QSYS.LIB 和 QOPT 文件系统中的对象。

缓冲的输入和输出

来自外部设备的输入以及到外部设备的输出在 OS/400 上进行缓冲；它由处理数据阻塞的输入和输出处理器处理。相反，UNIX 系统通常使用逐个字符（非缓冲）输入和输出的方式进行操作。在 OS/400 上，只有某些输入和输出信号（例如，执行键、功能键和系统需求）向系统发送中断。

数据转换支持

OS/400 PASE 程序将 ASCII（或 UTF-8）路径名传送给 `open` 函数来打开字节流文件，此时名称自动地转换为 OS/400 所使用的编码方案，但任何从打开文件中读取或写入的数据都未转换。

有关数据转换的更多信息，参见『数据编码』。

文件描述符的使用

OS/400 PASE 运行时通常将 ILE C 运行时支持用于文件 `stdin`、`stdout` 和 `stderr`，它们为 OS/400 PASE 和 ILE 程序提供一致的行为。

OS/400 PASE 和 ILE C 使用相同的标准输入和输出的流（stdin、stdout 和 stderr）。OS/400 PASE 程序始终访问使用描述符 0、1 和 2 的标准输入和输出。然而 ILE C 不总是将集成文件描述符用于 stdin、stdout 和 stderr，因此，OS/400 PASE 在 OS/400 PASE 文件描述符和集成文件系统中的描述符之间提供一个映射。因为此映射，OS/400 PASE 程序和 ILE C 程序可能使用不同的描述符号码来访问同一个打开文件。

可以将 OS/400 PASE 扩展 F_MAP_XPFFD 用在 `fcntl` 函数上，以将 OS/400 PASE 描述符分配给 ILE 号码。 如果 OS/400 PASE 应用程序需要为不是由 OS/400 PASE 创建的 ILE 描述符执行文件操作，则它是有用的。

`fstatx` 函数的 OS/400 专用的扩展 STX_XPFFD_PASE，允许 OS/400 PASE 程序为 OS/400 PASE 文件描述符确定集成文件系统描述符号码。为任何附加于 ILE C 运行时支持（用于文件 `stdin`、`stdout` 和 `stderr`）的 OS/400 PASE 描述符返回特殊值（负数）。

当调用 Qp2RunPase() API 时，如果 ILE 环境变量 QIBM_USE_DESCRIPTOR_STDIO 设置为 Y 或 I，则 OS/400 PASE 使文件描述符 0、1 和 2 与集成文件系统同步，这样，OS/400 PASE 和 ILE C 程序便对文件 stdin、stdout 和 stderr 使用相同的文件描述符号码。当以此方式操作时，如果 OS/400 PASE 代码或 ILE C 代码关闭或重新打开文件描述符 0、1 或 2，则更改影响两种环境下的 stdin、stdout 和 stderr 处理。

OS/400 PASE 运行时通常不执行通过 OS/400 PASE 文件描述符（包含套接字）读取或写入数据的字符编码转换，除非为从 ILE C stdin 中读取的数据或者写入 ILE C stdout 和 stderr 中的数据执行 ASCII 到 EBCDIC 的转换（OS/400 PASE CCSID 和作业缺省 CCSID 之间）。

两个环境变量控制 stdin、stdout 和 stderr 的自动转换:

- 通常应用的变量是 QIBM_USE_DESCRIPTOR_STDIO。当设置为 Y 时，ILE 运行时使用这些文件的文件描述符 0、1 或 2。
- 特定于 PASE 的环境变量是 QIBM_PASE_DESCRIPTOR_STDIO。值 B 表示二进制，T 表示文本。

如果 ILE 环境变量 QIBM_USE_DESCRIPTOR_STDIO 设置为 Y，并且 QIBM_PASE_DESCRIPTOR_STDIO 设置为 B（允许从 stdin 读取二进制数据，并且写入至 stdout 或 stderr），则禁用 OS/400 PASE stdin、stdout 和 stderr 的 ASCII 到 EBCDIC 的转换。QIBM_PASE_DESCRIPTOR_STDIO 的缺省值为表示文本的 T。此值引起从 EBCDIC 到 ASCII 的转换。

有关文件系统的更多信息，参见集成文件系统主题。

全球化

» 因为 OS/400 PASE 运行时是基于 AIX 运行时，所以 OS/400 PASE 程序可以使用 AIX 所支持的同样丰富的语言环境、字符串处理、数据和时间服务、消息编目和字符编码转换编程接口集合。 «

OS/400 PASE 支持 AIX 运行时上的接口，以管理应用程序使用的语言环境和执行区分语言环境的函数（如 ctype 和 strcoll），包括支持单字节和多字节字符编码。

OS/400 PASE 包括 AIX 语言环境的子集，它为许多国家或地区及使用工业标准编码（代码集为 ISO8859-x）、代码集 IBM-1250 和代码集 UTF-8 的语言提供支持。 » OS/400 PASE 以三种不同的方式提供对 Euro 的支持: IBM-1252 语言环境和 ISO 8859-15 语言环境（两者都使用单字节编码），以及 UTF-8 语言环境。 «

> **注**: OS/400 PASE 的语言环境支持独立于 ILE C 程序所使用的语言环境支持格式（对象类型为 *CLD 和 *LOCALE）。除了内部结构的不同，现有的 ILE C 程序附带语言环境支持 ASCII。

创建新的语言环境

OS/400 PASE 不附带创建新的语言环境的实用程序。然而，可以为带有 localedef 实用程序的 AIX 系统上的 OS/400 PASE 中的使用创建语言环境。

更改语言环境

当 OS/400 PASE 应用程序更改语言环境时，它通常还应该更改 OS/400 PASE CCSID（使用_SETCCSID 运行时函数）以匹配新语言环境的编码。这确保了任何字符数据接口自变量由 OS/400 PASE 运行时正确解释（当调用 EBCDIC 系统服务时，也可能被转换）。可以使用 cstoccsid 运行时函数以确定对应于代码集名称的 CCSID。

OS/400 PASE 运行时在由 OS/400 PASE 程序创建的任何文件上将 CCSID 标记设置为当前 OS/400 PASE CCSID 值（当程序开始或使用最近的 _SETCCSID 值时提供）。

应该使用支持日文、韩国语、繁体中文和简体中文的 OS/400 PASE 应用程序的 UTF-8 语言环境。OS/400 包括其它语言的语言环境，但系统不支持设置 OS/400 PASE CCSID 来匹配 IBM-euc*XX* 代码集的编码。当应用程序在其它平台上运行时，使用 UTF-8 支持可能需要转换以其它编码（如 Shift-JIS）存储的文件数据。

OS/400 PASE 转换对象和语言环境存储于何处

转换对象和 OS/400 PASE 的语言环境是与 OS/400 语言功能代码封装在一起的。当安装 OS/400 PASE 时，只能创建那些与已安装的 OS/400 语言功能相关的语言环境。

所有 OS/400 PASE 语言环境使用 ASCII 或 UTF-8 字符编码；因此，所有 OS/400 PASE 运行时以 ASCII（或 UTF-8）方式工作。

有关 OS/400 上的全球化的更多信息，参见全球化主题。

消息服务

OS/400 PASE 信号和 ILE 信号是独立的，因此，不可能通过产生信号的另一类型来直接调用某一信号类型的句柄。您可以使用 OS/400 PASE Qp2SignalPase() API 为接收的任何 ILE 信号发送相应的 OS/400 PASE 信号。QP2SHELL() 程序和 OS/400 PASE `fork()` 函数始终会建立句柄，以将每个 ILE 信号映射为相应的 OS/400 PASE 信号。

系统会自动将任何 OS/400 异常消息（已发送至某个调用的程序消息队列，该调用运行 Qp2RunPase、Qp2CallPase 或 Qp2CallPase2 API）转换为相应的 OS/400 PASE 信号。因此，OS/400 PASE 应用程序可以通过处理系统转换的 OS/400 PASE 信号来处理任何 OS/400 异常。

OS/400 PASE 提供了以下运行时函数，使您可以直接控制 OS/400 消息处理:

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

有关这些函数的详细信息，参见运行时函数。

OS/400 消息支持

OS/400 在多种上下文中提供消息支持:

作业记录

作业记录将包含所有由 OS/400 发出的或由应用程序在运行或编译时发出的消息。要查看记录，请在命令行中输入 DSPJOBLOG。在出现“显示作业记录”屏幕时，请按 F10 键，然后按 Shift + F6。这些键组合将显示“显示所有消息”屏幕，并开始显示最新的消息。要查看任何特定消息的详细信息，请将游标移至要了解详细信息的消息，然后按 F1 键。

处理活动作业
处理活动作业（WRKACTJOB）命令用于检查 OS/400 中的作业和作业堆栈。

有关 OS/400 中消息支持的更多信息，参见工作管理主题。

有关 OS/400 PASE 和 OS/400 消息的更多信息，参见 OS/400 PASE 信号处理。

从 OS/400 PASE 应用程序中打印输出


您可以使用 QShell Rfile 实用程序从 OS/400 PASE 外壳程序中读取或写入输出。

以下示例将流文件 mydoc.ps 的内容作为未转换 ASCII 数据写入假脱机打印机设备文件 QPRINT，然后使用 CL LPR 命令将假脱机文件发送至其它系统:

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

伪终端（PTY）


OS/400 PASE 支持 AT&T 和 Berkeley Software Distributions（BSD）样式的设备。从编程观点来看，这些设备在 OS/400 PASE 和 AIX 中以相同的方式工作。

对于 AT&T 样式的设备，OS/400 PASE 最多允许 1024 个实例，对于 BSD 样式的设备则最多允许 592 个实例。在启动系统时，将自动为每个设备类型创建前 32 个实例。

在 OS/400 PASE 中配置 PTY 设备

在 AIX 中，管理员可以使用 smit 来配置每种类型的可用设备数。在 OS/400 PASE 中，这些设备以如下方式配置:

- 对于 AT&T 样式的设备，OS/400 PASE 支持自动配置。如果前 32 个实例已在使用，而应用程序试图打开另一个实例，系统将在集成文件系统中自动创建 CHRSF 设备，最多可创建 1024 个设备。
- 对于 BSD 样式的设备，您必须使用 OS/400 PASE mknod 实用程序人工创建 CHRSF 设备。要完成此操作，您需要知道 BSD 从属设备和 BSD 主设备的最大数目以及命名约定。以下示例外壳程序脚本显示如何创建附加 BSD PTY 设备。该程序将创建 16 组设备。

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

prefix="pqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
bsd_tty_major=32949
bsd_pty_major=32948

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

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

while [ "$1" ]
```

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

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

  echo "creating /dev/[pt]ty${pre}0 - /dev/[pt]ty${pre}f"
  for i in 0 1 2 3 4 5 6 7 8 9 a b c d e f
  do
    echo ".\c"
    mkdev /dev/pty${pre}${i} $bsd_pty_major $minor
    echo ".\c"
    mkdev /dev/tty${pre}${i} $bsd_tty_major $minor
    minor=$((minor + 1))
  done
  echo ""

  shift
done
```

有关伪终端设备的更多信息，参见 AIX documentation  web 站点。 

安全性

出于安全性考虑，OS/400 PASE 程序与 OS/400 中的任何其它程序受相同的安全性限制。要在 OS/400 上运行 OS/400 PASE 程序，您必须具有访问集成文件系统中的 AIX 二进制的权限。您还必须对程序访问的每个资源具有适当级别的权限，否则在试图访问那些资源时，程序将接收到错误。

在运行 OS/400 PASE 程序时，以下信息特别重要。

用户概要文件和权限管理

系统权限管理基于也是对象的用户概要文件。系统中创建的所有对象均由某个特定用户拥有。系统将验证对某个对象的每次操作或访问，以确保用户的权限。所有者或适当的已授权的用户概要文件可以将不同类型的权限委托给其它用户概要文件，以对对象进行操作。系统将对所有类型的对象提供一致的权限检查。

对象权限机制提供了各种级别的控制。用户的权限受其所需限制。存储在 QOpenSys 文件系统中的文件的授权方式与 UNIX 文件相同。下表显示了 UNIX 权限和 OS/400 数据库文件中使用的安全性值之间的关系。在 OS/400 中，*OBJOPR 为*使用对象*权限；*EXCLUDE 为*无权限*。*READ、*ADD、*UPD、*DLT 和 *EXECUTE 为数据权限。要将文件作为 OS/400 PASE 程序运行，您需要为文件指定 *EXECUTE 权限（有时为 *READ 权限）。

UNIX 权限	*OBJOPR	*READ	*ADD	*UPD	*DLT	*EXECUTE
r（读）	X	X	-	-	-	-
w（写）	X	-	X	X	X	-
x（执行）	X	-	-	-	-	X
无权限	-	-	-	-	-	-

OS/400 PASE 中的用户概要文件

在 OS/400 中，认证信息存储在单独的*概要文件*中，而不是存储在类似于 /etc/passwd 的文件中。用户和组拥有概要文件。所有这些概要文件共享一个名称空间，而每个概要文件必须拥有唯一的单字符名称。如果将小写名称传送至 getpwnam() 或 getgrnam() API，系统会将名称字符串转换为所需的大小写形式。

» 如果调用 getpwuid() 或 getgrgid() 来获得返回的概要文件名，该名称将为小写，除非您将 OS/400 PASE 环境变量 PASE_USRGRP_LOWERCASE 设置为 N，该变量将以大写形式返回结果。 «

每个用户拥有用户标识（UID）。每个组拥有组标识（GID）。这些标识根据 POSIX 1003.1 标准定义。两个数字空间是独立的，因此您可以使用户拥有 UID 104 而使组拥有 GID 104，这两个标识互不相同。

对于安全性高级用户 QSECOFR，OS/400 拥有 UID 为 0 的用户概要文件。其它概要文件的 UID 不能为 0。QSECOFR 是系统中最有特权的概要文件，从这个意义上来说，QSECOFR 充当了 root 用户。但是，OS/400 还提供了一组特定权限，这些权限可以由系统管理员分配给单个用户。例如，其中一个特权 *ALLOBJ 优先于文件访问的任意访问控制，这是 UNIX 系统中 root 特权的典型使用。

在使用 root 访问的移植应用程序中，为可以授予 *ALLOBJ 权限的*应用程序用户*创建特定的用户概要文件是较好的安全性习惯。这样可以避免使用 QSECOFR，它具有比单一应用程序所需的特权更多的特权。与 UNIX 系统不同，OS/400 不需要用户的组成员资格。OS/400 中某个用户概要文件的 GID 0 表示*未分配组*，而不是指拥有更多特权的组。

OS/400 安全性依赖于系统中建立的集成安全性。所有对对象的访问必须通过安全性检查。就访问时为用户概要文件运行的进程而言，安全性检查已完成。

要维护完整性和安全性，OS/400 PASE 依赖于为每个进程指定单独的地址空间。如果某个资源在 OS/400 PASE 地址空间中不可用，则不能对其进行访问。文件系统安全性可以防止在未经适当授权的情况下将资源装入地址空间。资源一旦处于地址空间中就可用于进程，而不管进程运行所使用的标识。

OS/400 PASE 程序使用系统调用来请求系统函数。OS/400 PASE 程序的系统调用由 OS/400 处理。此接口使 OS/400 PASE 程序只能间接（并安全地）访问系统内部。

要了解 iSeries 服务上安全性的更多信息，参见安全性主题。

工作管理

OS/400 用与处理系统上其它作业相同的方法处理 OS/400 PASE 程序。有关 OS/400 如何处理作业的信息，参见工作管理主题。

对 OS/400 PASE 进行故障诊断

本信息提供有关如何调试 OS/400 PASE 程序及使其高效运行的指导:

- 调试 OS/400 PASE 程序
- 最优化性能

调试 OS/400 PASE 程序

OS/400 PASE 运行时环境为 syslog() 运行时函数提供库支持，并且为更多复杂的消息路由提供 syslogd 二进制。另外，可以使用 OS/400 中现有的功能，如诊断消息的作业记录和将服务器消息发送到 OS/400 系统操作程序消息队列 QSYSOPR 的作业记录。

调试 OS/400 PASE 应用程序的策略根据不同的应用程序可以使用不同的路径:

- 如果应用程序不要求任何 OS/400 集成（如，与 DB2 UDB iSeries 版或 ILE 功能集成），则应该首先在 AIX 上调试该应用程序。
- 然后，使用 OS/400 PASE dbx 和 OS/400 调试能力的组合（如作业记录）来在 OS/400 上调试应用程序。

已经编写的使用数据库或 ILE 函数的应用程序不能在 AIX 上进行全面测试，但可以在 AIX 上调试应用程序的其余部分，以保证它们的正确结构和设计。

在 OS/400 PASE 中使用 dbx

OS/400 PASE 支持 AIX dbx 调试器实用程序。如果实用程序已在源代码层上编译，则此程序允许在源代码层调试相关进程，如父进程和子进程。可以使用网络文件系统 (NFS) 使 AIX 源代码对于运行在 OS/400 PASE 上的调试器可见。

» OS/400 PASE 支持 xterm 和 aixterm，以允许使用 dbx 来调试父进程和子进程。dbx 启动另一个带有连接到第二个进程的 dbx 的 xterm 窗口。 «

有关 dbx 的详细信息，参见 AIX documentation Web 站点。也可以在 dbx 命令行输入 help。

使用 OS/400 调试工具

ILE C 源调试器是确定代码问题的有效工具。要学习关于此工具的使用，参见 WebSphere(TM) Development Studio ILE C/C++ Programmer's Guide 。

» 还可以使用 iSeries 系统调试器来调试 OS/400 PASE 代码。 «

优化性能

要获得最佳性能，请确保将应用程序二进制存储在本地流文件系统中。如果二进制（基本程序和库）在本地流文件系统之外，由于文件映射无法完成，因此启动 OS/400 PASE 程序会慢很多。

» 如果在执行大量 fork() 运算的 OS/400 PASE 中运行应用程序，该程序将不会象在 AIX 中运行那样快。这是因为每个 OS/400 PASE fork() 运算会启动一个新的 OS/400 作业，这样会对性能产生极大的影响。 «

有关收集和分析性能数据的信息，参见“系统管理”目录中的性能主题。

示例

» 以下示例已经在 OS/400 PASE 信息中有所提供。使用这些示例之前，请阅读以下的代码示例不保证声明。

从 ILE 程序运行 OS/400 PASE 程序和过程

- 第 17 页的『示例: OS/400 程序内运行 OS/400 PASE 程序』
- 第 19 页的『示例: 从 OS/400 程序内调用 OS/400 PASE 过程』

从 OS/400 PASE 程序中调用 OS/400 程序

- 第 21 页的『示例: 调用 ILE 过程』
- 第 28 页的『示例: 从 OS/400 PASE 调用 OS/400 程序』
- 第 30 页的『示例: 从 OS/400 PASE 运行 OS/400 命令』

使用 OS/400 PASE 程序中的 DB2 UDB iSeries 版函数

- 第 32 页的『示例: 在 OS/400 PASE 程序中调用 DB2 UDB iSeries 版 CLI 函数』

代码示例不保证声明

IBM 授予您非专用版权许可证来使用所有编程代码示例，您可以根据它们生成针对您的特定需要进行裁制的类似功能。

所有样本代码由 IBM 提供，仅供说明。这些示例并未在所有条件下进行完全测试。因此，IBM 不能保证或暗示这些程序的可靠性、可服务性或功能。

此处包含的所有程序均是以“按现状”的基础提供，不附有任何形式的保证。明确拒绝有关非侵权性、适销性和适用于某特定用途的默示保证。

有关 OS/400 PASE 的相关信息

有关如何使用 OS/400 PASE 的更多信息，参见下列资源。

其它 iSeries 信息中心主题

OS/400 PASE API

有关以下一般类别的 OS/400 PASE API 的详细信息，参见本主题:

- 可调用的程序 API
- ILE 过程 API
- 由 OS/400 PASE 程序所使用的运行时函数

您必须调用系统 API 来运行 OS/400 PASE 程序。系统提供可调用的程序 API 和 ILE 过程 API 来运行 OS/400 PASE 程序。可调用的程序 API 更为容易使用，但是它却未能提供 ILE 过程 API 的所有可用控件。

OS/400 PASE 外壳程序和实用程序

OS/400 PASE 包括三个外壳程序（Korn、Bourne 与 C 外壳程序）和将近 200 个作为 OS/400 PASE 程序运行的实用程序。OS/400 PASE 外壳程序和实用程序提供可扩展的脚本编制环境（包括大量的业界标准和实际标准命令）。

OS/400 PASE 命令

本主题中描述的大多数 OS/400 PASE 命令支持与 AIX 命令相同的选项并提供与其相同的行为。除了 OS/400 PASE 命令，每个 OS/400 PASE 外壳程序均支持许多内置命令（如 `cd`、`exec` 和 `if`）。

OS/400 PASE 库

OS/400 PASE 运行时支持由 AIX 运行时提供的接口的大子集。OS/400 PASE 所支持的大多数运行时接口提供与 AIX 相同的选项和行为。OS/400 PASE 运行时库（作为符号链接）安装在 `/usr/lib` 中。

Web 站点

OS/400 PASE 支持的 AIX 命令、API 和实用程序: API Analysis Tool  是查找有关 OS/400 PASE 是如何支持应用程序使用 AIX 函数的详细信息的最有效、高效的方法。

IBM PartnerWorld for Developers Web 站点中的 Application Factory  和 OS/400 PASE  页提供了有关将应用程序移植到使用 OS/400 PASE 的 iSeries 服务器的一般信息。Application Factory 将 OS/400 PASE 与用于将应用程序移植到 iSeries 服务器的其它解决方案相比较。

有关 AIX 命令和实用程序的更多信息，参见 AIX documentation  Web 站点。

中国印刷