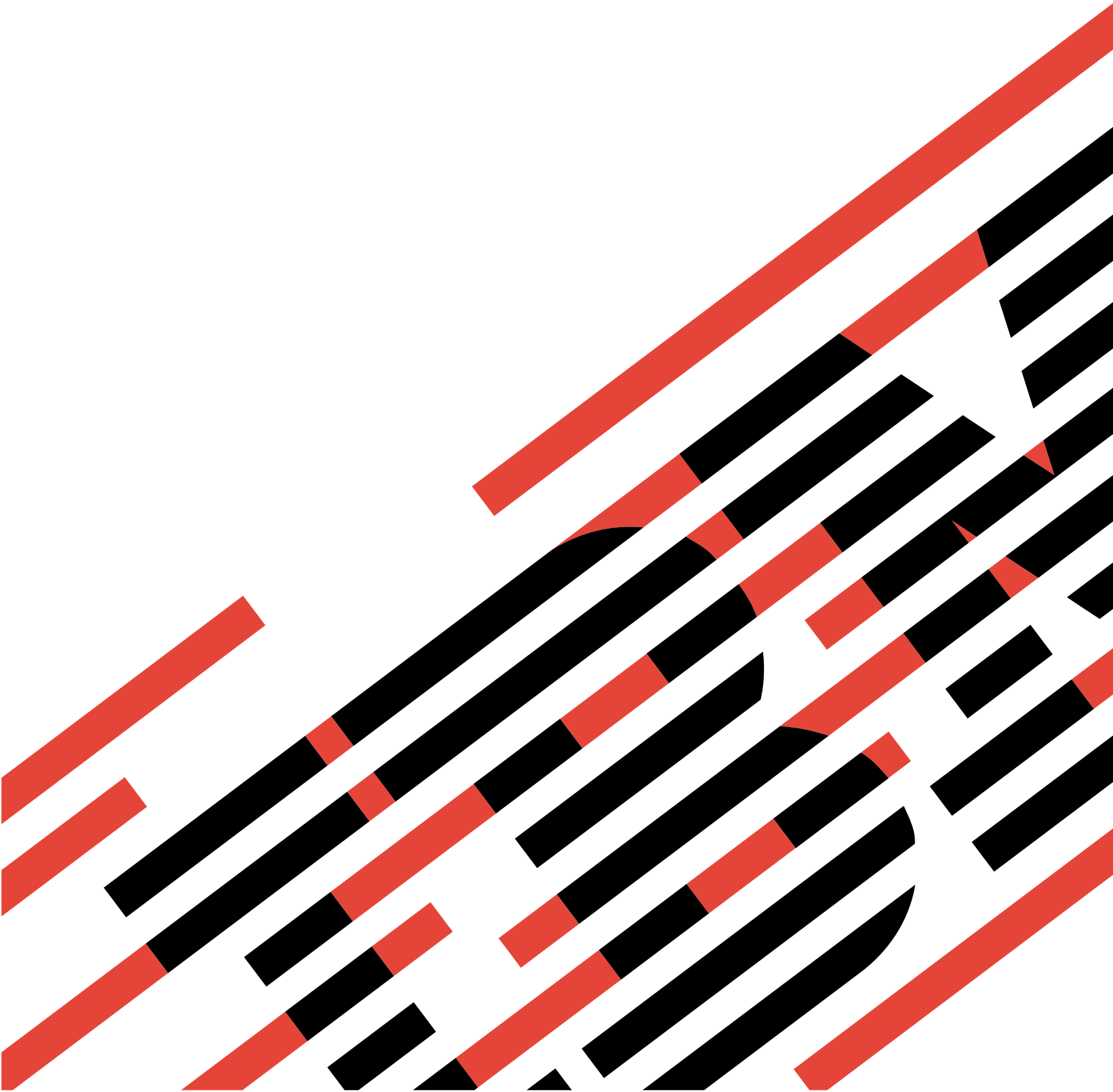
IBM

eserver

iSeries

OS/400 PASE



IBM

@server

iSeries

OS/400 PASE

목차

OS/400 PASE

OS/400(R) PASE(OS/400 Portable Application Solutions Environment)를 사용하면 최소의 노력으로 AIX(R) 어플리케이션을 iSeries(TM) 서버에 이식할 수 있습니다. OS/400 PASE는 복잡한 UNIX 시스템 관리를 수행하지 않고서도 선택된 UNIX(R) 어플리케이션을 실행할 수 있는 통합 런타임 환경을 제공합니다.

OS/400 PASE는 또한 강력한 스크립팅 환경을 제공하는 산업 표준이며 실질적 표준인 쉘과 유틸리티도 제공합니다.

OS/400 PASE에 좀 더 익숙해지려면 다음을 참조하십시오. 이 릴리스의 새로운 사항에 대한 정보와 이 주제 인쇄 방법에 대한 정보를 찾을 수 있습니다.

OS/400 PASE 시작하기
OS/400 PASE(Portable Application Solutions Environment)의 개요를 제공하고 OS/400 PASE가 도움이 될 수 있는 때와 방법을 설명하며 iSeries 서버에 OS/400 PASE를 설치하기 위한 지침을 제공합니다.

OS/400 PASE 계획
OS/400 PASE를 사용하기 전에 고려해야 하는 기술적 요구사항에 대해 논의합니다.

OS/400 PASE에서 실행할 프로그램 준비
OS/400 PASE에서 효과적으로 실행되는 AIX 프로그램의 작성, 컴파일 및 복사 지침을 제공합니다.

OS/400 환경에서 OS/400 PASE 프로그램 사용
OS/400 환경에서 OS/400 PASE를 실행하는 방법, OS/400 PASE 프로그램 내에서 OS/400 프로그램과 ILE 프로시듀어를 호출하는 방법, OS/400 PASE 프로그램이 OS/400 기능(예: 보안, 메세지, 데이터베이스, 통신, 작업 관리, 인쇄 및 통합 언어 환경(R)(ILE)과 대화하는 방식에 대해 논의합니다.

OS/400 PASE 문제 해결
OS/400 PASE 프로그램의 오류를 정정하고 이 프로그램을 좀 더 효율적이며 효과적으로 실행하기 위한 정보를 제공합니다.

예
이 정보의 각 예와 예를 사용하기 전에 읽어야 하는 코드 예 면책사항에 대한 링크를 제공합니다.

OS/400 PASE 관련 정보
OS/400 PASE API 라이브러리 및 유틸리티에 대한 자세한 정보를 찾을 수 있는 iSeries Information Center 내의 위치를 알려줍니다. OS/400 PASE 및 AIX와 관련하여 Information Center 외부의 추가 정보에 대한 링크를 제공합니다.

V5R2의 새로운 사항

다음은 V2R2에서의 OS/400 PASE에 대한 몇가지 중요한 개선점 및 변경사항입니다.

- OS/400 PASE는 OS/400 PASE 환경에서 두 개의 AIX 컴파일러 제품 중 하나를 설치하고 AIX에서 OS/400 PASE 프로그램을 테스트 및 컴파일하기 위해 요구 제거를 지원합니다. 자세한 내용은 AIX 소스 컴파일을 참조하십시오.
- 유사 단말기(PTY) 지원 및 UNIX 스타일 작업 제어. 자세한 내용은 유사 단말기(PTY)를 참조하십시오.
- 100개 이상의 새 유틸리티. 전체 리스트를 보려면 OS/400 PASE 쉘 및 유틸리티 주제를 참조하십시오.
- 다음과 같은 새 라이브러리가 추가되었습니다. 전체 리스트를 보려면 OS/400 PASE 런타임 라이브러리 주제를 참조하십시오.

libcur.a	AIX legacy Curses 라이브러리
libg.a	디버그 지원
libgair4.a	내부 X Windows 지원
libl.a	lex 지원
libld.a	오브젝트 파일 액세스 루틴 라이브러리
libm.a	IEEE Math 라이브러리
libPW.a	프로그래머 워크벤치 라이브러리
libxcurses.a	Curses 라이브러리
libXi.a	X Windows 입력 처리
libXtst.a	X Windows 테스트 지원
liby.a	yacc 지원

- OS/400 PASE 프로그램에서 사용하는 지원되는 않는 시스템 호출에 대해 새 메세지(MCH3204)가 작업 기록부에 표시됩니다. 이 메세지 텍스트에는 시스템 호출의 이름과 오류를 발생한 명령어 주소가 들어 있습니다.
- 신규 및 변경된 OS/400 PASE 런타임 함수:
  - _CVTERRNO(OS/400 PASE 오류 번호를 ILE 오류 번호로 변환)
  - _ILECALLX(고급 ILE 프로시듀어 호출)
  - _PMGCALL(OS/400 프로그램 호출)
  - _RETURN(기존 OS/400 PASE 없이 리턴)
  - _RSLOBJ, _RSLOBJ2(OS/400 오브젝트로 변형)
  - _STRLEN_SPP, _STRCPY_SPP(16바이트 ILE 포인터를 사용한 스트링 조작)
  - Qp2paseCCSID(OS/400 PASE CCSID 검색)
  - Qp2jobCCSID(마지막 OS/400 PASE CCSID 세트에서 작업 디폴트 CCSID 검색)
  - faccessx
  - fchdir
  - fclear
  - fclear
  - getaddrinfo, getnameinfo

- getcontext, setcontext
- getpri, getpriority, setpriority
- getprocs64, getthrds64
- gettimer, settimer
- msem_init, msem_lock, msleep, msem_unlock, msem_remove
- pread, pwrite
- setgroups
- sigstack, sigaltstack(대체 신호 스택)
- statpriv
- statvfs, fstatvfs
- sync
- ustat

• 신규 및 변경된(ILE) OS/400 PASE용 API:

- QP2SHELL2(QP2SHELL과 유사하지만 호출자의 활성 그룹에서 실행됩니다.)
- Qp2ptrsize(OS/400 PASE 포인터 크기 검색)
- Qp2paseCCSID(OS/400 PASE CCSID 검색)
- Qp2jobCCSID(OS/400 PASE CCSID 마지막 세트에서 작업 디폴트 CCSID 검색)
- Qp2errnop(현재 스레드의 OS/400 PASE 오류 번호 찾기)
- Qp2malloc(OS/400 PASE 힙(heap) 메모리 할당)
- Qp2free(OS/400 PASE 힙(heap) 메모리 비우기)
- Qp2dlopen(OS/400 PASE 모듈 동적 로드)
- Qp2dlsym(OS/400 PASE dlopen이 연 모듈에서 기호 찾기)
- Qp2dlclose(OS/400 PASE dlopen이 로드한 모듈을 닫고 언로드)
- Qp2dlerror(마지막 동적 로드 조작의 오류 정보 검색)
- Qp2CallPase(및 Qp2CallPase2) 개선: by-address 인수와 결과 및 OS/400 PASE가 시작하지 않은 스레드에서 OS/400 PASE 프로시듀어 호출

• OS/400 PASE 로케일(및 국제화 지원을 위한 기타 파일)은 OS/400 언어 피처 코드와 함께 들어 있습니다. 자세한 내용은 국제화를 참조하십시오. 또한 서로 다른 키보드와 문자 세트를 처리하는 X Windows를 위한 200개 이상의 새 파일과 다음의 65개 신규 로케일이 추가되었습니다. 전체 리스트를 보려면 OS/400 PASE 로케일 주제를 참조하십시오.

```
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
```

새로운 사항과 변경된 사항을 보는 방법

어디에서 기술적 변경이 이루어졌는지 쉽게 확인할 수 있도록 이 정보에서는 다음을 사용합니다.

- 
  새 정보나 변경된 정보가 시작되는 위치를 표시하는 이미지
- 
  새 정보나 변경된 정보가 끝나는 위치를 표시하는 이미지

이 릴리스에서 새로운 사항 또는 변경 사항에 대한 더 많은 정보를 보려면 사용자 메모  를 참조하십시오.

이 주제 인쇄

PDF 버전을 보거나 다운로드하려면 OS/400 PASE 정보(약 211KB 또는 52 페이지)를 선택하십시오.

PDF 파일 저장

PDF를 보거나 인쇄를 위해 워크스테이션에 저장하려면 다음을 수행하십시오.

1. 브라우저에서 PDF를 마우스 오른쪽 단추로 클릭하십시오(위의 링크를 마우스 오른쪽 단추로 누르십시오).
2. **다른 이름으로 대상 저장...**을 클릭하십시오.
3. PDF를 저장하려는 디렉토리로 가십시오.
4. **저장**을 클릭하십시오.

Adobe Acrobat Reader 다운로드

PDF를 보거나 인쇄하기 위해 Adobe Acrobat Reader가 필요한 경우 Adobe 웹 사이트 (www.adobe.com/products/acrobat/readstep.html)  에서 사본을 다운로드하십시오.

OS/400 PASE 시작하기

교차 플랫폼 어플리케이션의 개발과 전개는 효율적 비즈니스 컴퓨팅 환경의 중요 구성요소입니다. 또한 사용자 시스템이 제공하는 기능의 통합과 사용의 용이성도 마찬가지로 중요합니다. 이 점이 바로 iSeries 및 AS/400e(TM) 서버의 특징이기도 합니다. 비즈니스가 점차 개방형 컴퓨팅 환경으로 이동해감에 따라 서로 상이하기 까지한 이러한 목표를 달성하는 것이 어렵고, 시간 소모가 많으며 고비용의 작업이라는 것을 알게 될 것입니다. 예를 들어 실행 중인 유사 어플리케이션을 활용하면서 AIX 오퍼레이팅 시스템의 기능을 활용하고자 하지만 AIX와 OS/400 오퍼레이팅 시스템 모두를 관리해야 하는 부담은 원하지 않을 것입니다.

이런 경우 OS/400 PASE(Portable Application Solutions Environment)가 도움이 됩니다. OS/400 PASE를 이용하면 최소한의 변경으로 또는 전혀 변경하지 않고도 OS/400에서 다수의 AIX 어플리케이션 2진을 실행할 수 있어서 플랫폼 솔루션 포트폴리오를 효과적으로 확장해줍니다.

OS/400 PASE에 대해 더 자세히 알려면 다음 주제를 참조하십시오.

- OS/400 PASE 개념
- 어플리케이션 개발에서 OS/400 PASE가 유용한 옵션이 되는 경우
- OS/400 PASE 설치

OS/400 PASE 개념

OS/400 PASE(OS/400 Portable Application Solutions Environment)는 OS/400에서 실행 중인 AIX 어플리케이션을 위한 통합된 런타임 환경입니다. AIX의 ABI(Application Binary Interface)를 지원하고 AIX 공유 라이브러리, 쉘 및 유틸리티가 지원하는 광범위한 서브세트를 제공합니다. OS/400 PASE는 PowerPC(TM) 기계 명령어의 직접 실행을 지원하므로 기계 명령어만을 에뮬레이트하는 환경이 갖는 약점이 없습니다.

OS/400 PASE 어플리케이션:

- C, C++, Fortran 또는 PowerPC 어셈블러로 작성할 수 있습니다.
- AIX PowerPC 어플리케이션과 동일한 2진(binary) 실행 형식을 사용합니다.
- OS/400 작업에서 실행됩니다.
- 파일 시스템, 보안 및 소켓과 같은 OS/400 시스템 기능을 사용합니다.

OS/400 PASE는 OS/400의 UNIX 오퍼레이팅 시스템이 아닙니다. OS/400 PASE는 최소한의 변경으로 OS/400에서 AIX 프로그램을 실행하도록 설계되었습니다. 다른 UNIX 기반 환경의 프로그램은 OS/400 PASE에서 실행하기 위한 첫 번째 단계로서 AIX에서 컴파일되도록 작성해야 합니다.

OS/400 PASE 통합 런타임은 iSeries 서버에서 사용권 내부 코드(LIC) 커널로 실행됩니다. 시스템은 OS/400 PASE에 있는 다수의 일반적인 OS/400 기능과 다른 런타임 환경(ILE 및 Java(TM))을 통합합니다. OS/400 PASE는 AIX 시스템 호출의 광범위한 서브세트를 구현합니다.

OS/400 PASE에 대한 시스템 지원은 OS/400 PASE 프로그램이 액세스할 수 있는 메모리를 제어하고 권한이 없는 기계 명령어만 사용하도록 제한하여 시스템 보안과 무결성을 보장합니다.

«

최소의 노력으로 신속한 어플리케이션

많은 경우에 AIX 프로그램은 최소한의 변경으로 또는 전혀 변경하지 않고도 OS/400 PASE에서 실행됩니다. 필요한 AIX 프로그래밍 기술 수준은 AIX 프로그램의 설계에 따라 다릅니다. 또한 사용자의 프로그램 설계(가령 CL 명령)에 추가의 OS/400 어플리케이션 통합을 제공하여 어플리케이션 사용자들의 구성 노력을 최소화할 수 있습니다.

OS/400 PASE는 OS/400 시장에서의 성공을 공유하고자 하는 솔루션 개발자들을 위해 또다른 이식 옵션을 추가합니다. OS/400 PASE는 이식 시간을 크게 줄일 수 있는 수단을 제공하여 출시까지의 시간을 단축하고 솔루션 개발자의 투자 수익을 증가시킵니다.

OS/400에서의 광범위한 AIX 기술 서브세트

OS/400 PASE는 다음과 같은 광범위한 AIX 기술 서브세트에 기반한 어플리케이션 런타임을 구현합니다.

- 표준 C 및 C++ 런타임(스레드세이프 및 비스레드세이프)
- Fortran 런타임(스레드세이프 및 비스레드세이프)
- `pthreads` 스레드 패키지
- `iconv` 서비스(자료 변환용)
- BSD(Berkeley Software Distributions)에 상당하는 지원
- X-windows 클라이언트 지원(Motif 위젯 세트 포함)
- »
  유사 단말기(PTY) 지원

  «

어플리케이션은 OS/400 PASE가 지원하는 레벨과 호환하는 AIX 레벨을 실행 중인 AIX 워크스테이션에서 개발 및 컴파일되어 OS/400에서 실행됩니다.

»

대안으로, OS/400 PASE 안에서 어플리케이션을 개발, 컴파일, 빌드 및 실행하기 위해 OS/400 PASE 환경에 다음 제품 중 선택하여 설치할 수 있습니다.

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

자세한 내용은 AIX 소스 컴파일을 참조하십시오.

OS/400 PASE에는 강력한 스크립팅 환경을 제공하는 약 200개의 유틸리티와 Korn, Bourne 및 C 쉘도 들어 있습니다. 자세한 정보는 OS/400 PASE 쉘 및 유틸리티를 참조하십시오.

OS/400 PASE는 AIX 및 OS/400 오퍼레이팅 시스템에 대해 IBM investment의 일반 프로세서 기술을 사용합니다. PowerPC 프로세서는 OS/400 PASE 런타임에서 어플리케이션을 실행하기 위해 OS/400 모드에서 AIX 모드로 전환합니다.

OS/400 PASE에서 실행 중인 어플리케이션은 OS/400 통합 파일 시스템 및 iSeries용 DB2(R) Universal Database(R)와 통합됩니다. 이들 어플리케이션은 Java 및 통합 언어 환경(ILE) 어플리케이션을 호출할 수 있고 그 역도 가능합니다. 보통 이들 어플리케이션은 OS/400 조작 환경의 모든 측면 즉 보안, 메세지, 통신, 백업 및 회복 등을 활용할 수 있습니다. 동시에 AIX 인터페이스에서 파생된 어플리케이션 인터페이스도 활용합니다.

어플리케이션 개발에서 OS/400 PASE가 유용한 경우

OS/400 PASE는 AIX 어플리케이션을 iSeries 서버에 이식하는 방식을 결정할 때 상당한 유연성을 제공합니다. 물론 OS/400 PASE는 선택 가능한 여러 옵션 중 하나입니다.

API 분석

어플리케이션이 OS/400 PASE에 적합한가를 판별하는 데 있어서의 시작점은 API, 라이브러리 및 사용하는 유틸리티와 OS/400에서의 실행 효율 등과 같은 어플리케이션 분석입니다. IBM PartnerWorld(TM)

팀은 어플리케이션을 분석하고 잠재적 난제들을 기술하는 무료 이식 견적 툴인 API 분석 툴

을 사용하여 이러한 부분에서 도움을 제공합니다. 어플리케이션을 OS/400 PASE에 이식하는 프로시듀어에 대해 분석 툴의 적합 여부를 판별하는데 대한 자세한 정보는 OS/400 PASE에서 실행할 프로그램 준비를 참조하십시오.

잠재적 OS/400 PASE 어플리케이션의 특성

다음은 OS/400 PASE의 사용 여부에 대한 의사 결정을 할 때 고려할 몇 가지 유용한 지침입니다.

- **AIX 어플리케이션이 계산 집약적인가?**
  OS/400 PASE는 고도로 최적화된 연산 라이브러리를 제공하여 iSeries 서버에서 계산 집약 어플리케이션을 실행하기 위한 좋은 환경을 제공합니다.
- **어플리케이션이 OS/400 PASE에서만 지원되는(또는 ILE에서 부분적으로만 지원되는) 기능(예: `fork()`,**

X-Windows 또는 유사 단말기(PTY) 지원)에 의존하는 비중이 높은가?

OS/400 PASE는 fork() 및 exec()(현재 OS/400 시스템에 존재하지는 않음)에 대한 지원을 제공합니다(다만 spawn()의 사용은 제외되는데, 이는 fork() 함수를 exec() 함수와 통합합니다).

- **어플리케이션이 복잡한 AIX 기반 빌드 프로세스나 테스트 환경을 사용하는가?**
  OS/400 PASE를 이용하면 새 플랫폼으로 바로 전송할 수 없는 기존의 복잡한 프로세스를 가지고 있을 경우 특히 유용한 AIX 기반 빌드 프로세스를 사용할 수 있습니다.
- **어플리케이션이 ASCII 문자 세트에 대해 종속성을 갖는가?**
  OS/400 PASE는 이러한 필요성을 갖는 어플리케이션에 좋은 지원을 제공합니다.
- 
  어플리케이션이 다수의 포인터 조작을 수행하는가 또는 정수를 포인터로 변환(캐스트)하는가?
  OS/400 PASE는 최소의 성능 비용과 정수를 포인터로 변환하는 기능으로 32비트 및 64비트 AIX 주소지정 모델을 모두 지원합니다.

OS/400 PASE가 최적의 솔루션이 아닌 경우

OS/400 PASE는 ILE에서 호출해야 하는 다량의 호출가능 인터페이스를 제공하며 다음과 같은 특성을 갖는 코드에는 적합하지 않습니다.

- 
  각 호출 시마다 OS/400 PASE를 시작 또는 종료하거나 이미 사용 중인 프로그램에서(Qp2CallPase API를 사용하여) OS/400 PASE 프로시듀어를 호출하여 제공되는 것보다 고성능 호출과 리턴을 필요로 하는 코드

- ILE 호출자와 라이브러리 코드 사이에서 메모리나 이름공간을 공유해야 하는 코드. OS/400 PASE 프로그램은 자신을 호출한 ILE 코드와 메모리나 이름공간을 묵시적으로 공유하지 않습니다. (그러나 OS/400 PASE에서 호출된 ILE 코드는 OS/400 PASE 메모리를 공유하거나 사용할 수 있습니다.)

OS/400 PASE 설치

OS/400 PASE는 모든 iSeries 서버에서 무료로 사용할 수 있습니다. OS/400 PASE를 설치할 것을 권장합니다. 일부 시스템 소프트웨어(예: 고급 DNS(Domain Name Server) 서버 및 ILE C++ 컴파일러)에는 OS/400 PASE 지원이 필요합니다.

서버에 OS/400 PASE를 설치하려면 다음을 수행하십시오.

1. OS/400 명령행에서 GO LICPGM을 입력하십시오.
2. 11. 사용권 프로그램 설치를 선택하십시오.
3. 옵션 33(5722SS1 - Portable Application Solutions Environment)을 선택하십시오.

»

로케일 설치

OS/400 PASE 제품은 OS/400에 설치한 언어 피처와 연관된 로케일 오브젝트만 설치합니다. 언어 피처와 연관되지 않은 로케일이 사용자 서버에 필요한 경우 추가의 OS/400 언어 피처를 주문하여 설치해야 합니다. 자세한 내용은 OS/400 PASE 국제화 및 OS/400 PASE 로케일을 참조하십시오.

«

어플리케이션을 OS/400 PASE에 이식하는 소프트웨어 개발자를 위한 사용권 부여 정보:

OS/400 PASE는 OS/400 시스템에 AIX 런타임 라이브러리의 서브세트를 제공합니다. OS/400 사용권은 OS/400에 들어 있는 모든 라이브러리 코드를 사용할 수 있는 권한을 부여합니다. 이 사용권은 OS/400 PASE에 들어 있지 않은 AIX 라이브러리에 대한 사용권을 의미하지는 않습니다. 모든 AIX 제품은 IBM에서 별도로 사용권을 부여합니다.

사용자 어플리케이션을 OS/400 PASE에 이식하기 시작하면서 OS/400 PASE에 제공되지 않은 AIX 라이브러리에 대해 사용자 어플리케이션이 종속성을 가짐을 발견할 수도 있습니다. 이러한 라이브러리를 OS/400 시스템에 이식하기 전에 이들 라이브러리에 제공된 소프트웨어 제품을 판별하고 해당 소프트웨어 제품에 대한 사용권 계약의 조건을 검토해야 합니다. IBM 또는 제3자와 협력하여 추가의 미들웨어 종속 제품을 OS/400 시스템에 이식해야 할 수도 있습니다. 이식을 시작하기 전에 이식 중인 코드와 관련한 모든 사용권 계약을 면밀히 검토해야 합니다. IBM 소유의 라이브러리에 대해 적절한 사용권 계약 정보를 찾으려면 IBM 영업대표, IBM 이식 센터 중 하나, Rochester 소재의 고객 기술 센터 또는 개발자용 PartnerWorld에 문의하십시오.

OS/400 PASE 계획

OS/400 PASE는 최소한의 노력으로 AIX 어플리케이션을 iSeries 서버에 이식할 수 있도록 하는 AIX 런타임 환경을 OS/400에 제공합니다. 사실 많은 AIX 프로그램이 아무런 변경 없이 OS/400 PASE에서 실행됩니다. 이것은 OS/400 PASE가 AIX에서 사용할 수 있는 것과 같은 공유 라이브러리를 다수 제공하고 pSeries(TM) AIX PowerPC 프로세서에서와 같은 방식으로 iSeries PowerPC 프로세서에서 바로 실행되는 광범위한 AIX 유틸리티 서브세트를 제공하기 때문입니다.

OS/400 PASE에 대한 작업을 시작할 때 다음 사항에 유의하십시오.

- »

  AIX 2진의 목표 릴리스와 2진이 실행되는 OS/400 PASE 릴리스 사이에 상관이 존재합니다.

  AIX에서 OS/400 PASE 어플리케이션을 컴파일하는 경우 AIX에 작성되는 어플리케이션 2진은 어플리케이션을 실행하려는 OS/400 PASE 버전과 호환 가능해야 합니다. 다음 표는 OS/400 PASE의 여러 버전과

호환하는 AIX 2진 버전을 보여줍니다. 예를 들어 AIX 릴리스 4.3용으로 작성된 32비트 어플리케이션은 OS/400 PASE V4R4가 아니라 OS/400 PASE V4R5, V5R1 또는 V5R2에서 실행됩니다. 마찬가지로 AIX 릴리스 4.3용으로 작성된 64비트 어플리케이션은 OS/400 PASE V4R4, V4R5 또는 V5R2가 아니라 OS/400 PASE V5R1에서 실행됩니다.

«

AIX 릴리스	OS/400 V4R4	OS/400 V4R5	OS/400 V5R1	OS/400 V5R2
4.2(32비트)	X	X	X	X
4.3(32비트)	-	X	X	X
4.3(64비트)	-	-	X	-
5.1(32비트 또는 64비트)	-	-	-	X(주 참조)

»

주: 64비트 버전 5 릴리스 2 OS/400 PASE 어플리케이션은 AIX 5L(TM) 릴리스 5.1에서 재컴파일해야 합니다. 자세한 내용은 OS/400 PASE에서 실행할 프로그램 준비를 참조하십시오.

«

- **OS/400 PASE는 OS/400에서 AIX 커널을 제공하지 않습니다.**

  대신 공유 라이브러리가 필요로 하는 하위 시스템 함수가 OS/400 커널이나 통합 OS/400 함수에 라우트됩니다. 이 점과 관련하여 OS/400 PASE는 AIX와 OS/400 플랫폼 사이의 간격을 연결합니다. 사용자 코드는 공유 라이브러리의 API에 대해 AIX에서와 동일한 구문을 사용하지만 OS/400 PASE 프로그램은 OS/400 작업 내에서 실행되며 다른 OS/400 작업처럼 OS/400이 관리합니다.
- **대부분의 경우 OS/400 PASE에서 호출하는 API는 AIX에서와 정확하게 같은 방식으로 작동합니다.**

  그러나 일부 API는 OS/400 PASE에서 다르게 작동하며 OS/400 PASE에서 지원되지 않을 수도 있습니다. 이 때문에 OS/400 PASE 프로그램 준비에 대한 계획은 API 분석 툴을 사용한 전면적 코드 분석으로 시작해야 합니다. 이 툴은 AIX 어플리케이션을 OS/400 PASE에 이식할 때 고려해야 하는 프로그램 수정 유형을 포괄적으로 요약하여 제공합니다.
- **AIX와 OS/400 플랫폼 사이에 존재하는 차이점:**
  - AIX는 보통 대소문자를 구분하지만 특정 OS/400 파일 시스템은 그렇지 않습니다.
  - AIX는 보통 자료 코드화에 ASCII를 사용하지만 OS/400은 EBCDIC을 사용합니다. OS/400 PASE 프로그램에서 ILE(Integrated Language Environment) 코드의 세부사항을 관리할 경우 이 점을 고려해야 합니다. 예를 들어 OS/400 PASE에서 임의의 ILE 프로시듀어로 호출하는 경우 스트링에서 숫자 코드화 변환을 처리하도록 OS/400 PASE 프로그램을 명시적으로 코딩해야 합니다. OS/400 PASE 런타임 지원에는 문자 코드화 변환을 위한 `iconv_open()`, `iconv()` 및 `iconv_close()` 함수가 포함됩니다.

    주: OS/400 PASE 및 ILE은 자체의 변환표를 사용하여 `iconv()` 인터페이스를 독립적으로 구현합니다. OS/400 PASE `iconv()` 지원이 지원하는 변환은 통합 파일 시스템에 바이트 스트림 파일로 저장되기 때문에 사용자가 수정 및 확장할 수 있습니다.

- AIX 어플리케이션에서는(파일 및 쉘 스크립트에 있는) 행이 라인 피드(LF)로 끝나야 하지만 퍼스널 컴퓨터(PC) 소프트웨어 OS/400 소프트웨어는 보통 캐리지 리턴 및 라인 피드(CRLF)로 행을 끝냅니다.
- AIX에서 사용하는 일부 스크립트와 프로그램은 표준 유틸리티에 대해 고정된 경로를 사용할 수 있으므로 OS/400 PASE에서 사용하게 될 경로를 반영하도록 경로를 수정해야 합니다. 자세한 내용은 사용자 프로그램과 OS/400 PASE의 호환성 분석을 참조하십시오.

OS/400 PASE는 이러한 사안 중 일부를 자동으로 처리합니다. 예를 들어 시스템이 제공하는 OS/400 PASE 런타임 서비스(OS/400 옵션 33에 제공되는 공유 라이브러리에서의 시스템 호출 또는 런타임 함수)를 사용할 경우 OS/400 PASE는 필요한 대로 ASCII 대 EBCDIC 변환을 수행하는 반면 파일 설명자(바이트 스트림 파일 또는 소켓)에서 읽거나 여기에 기록한 자료에 대해서는 변환이 수행되지 않습니다.

다른 하위 함수(예: _ILECALL)를 사용하여 ILE 함수와 API에 대한 호출로 OS/400 PASE 프로그램의 기능성을 확장할 수 있습니다. 그러나 위에 언급한 대로 자료 변환을 처리해야 할 수 있습니다. 또한 프로그램에서 이와 같은 확장을 코딩하려면 추가의 헤더와 내보내기 파일을 사용해야 합니다.

OS/400 PASE에서 실행할 프로그램 준비

OS/400에서 효율적으로 실행되는 AIX 프로그램을 준비하기 위해 취해야 하는 단계는 프로그램의 특성과 OS/400 고유 인터페이스 및 기능을 사용해야 하는지의 여부에 따라 다릅니다.

UNIX 어플리케이션을 OS/400 PASE에 이식하려는 경우, 먼저 어플리케이션이

»

AIX 컴파일러

«

를 사용하여 컴파일 하는지 확인해야 합니다. 어떤 경우에는 이를 위해 UNIX 프로그램을 수정해야 합니다.

»

주: OS/400 PASE V5R2는 AIX 5L 릴리스 5.1을 지원합니다. OS/400 PASE에서 실행하고자 하는 새로운 64비트 어플리케이션은 AIX 5L 릴리스 5.1에서 컴파일해야 하며 기존의 64비트 어플리케이션은 재컴파일해야 합니다. OS/400 PASE는 재컴파일 없이 이전 AIX 릴리스의 32비트 어플리케이션을 지원합니다.

«

1단계: 프로그램 분석
이 프로세스의 첫 단계는 모든 경우에 수행이 권장됩니다. API 분석 툴을 사용하여 프로그램이 사용하는 API 및 OS/400 PASE에서 이들 API가 수행되는 방식에 대한 상세 보고서를 입수하십시오.

2단계: AIX 소스 프로그램 컴파일
사용자 프로그램이 OS/400 PASE 프로그램으로서 어느 정도 적합한지 판별하고 OS/400 PASE에서의

실행을 위해 필요한 수정을 가한 후 소스를 컴파일하십시오. (AIX 프로그램에 대한 분석 결과 OS/400 PASE에서 실행하기 위해 별도의 변경이 필요하지 않은 것으로 판명되면 프로그램을 재컴파일할 필요가 없습니다.)

»

OS/400 PASE 프로그램을 컴파일하기 위해 AIX 시스템을 사용하거나, OS/400 PASE 환경에서 프로그램을 컴파일하기 위해서 선택적으로 OS/400 PASE의 두 개의 AIX 컴파일러 제품 중 하나를 설치할 수 있습니다.

«

3단계: 프로그램을 iSeries 서버에 복사

»

AIX 시스템에서 OS/400 PASE 프로그램을 컴파일한 경우 2진 파일을 iSeries 서버에 복사하십시오.

«

4단계: 선택적으로 OS/400 인터페이스를 사용하도록 AIX 어플리케이션 사용자 정의
OS/400 고유 인터페이스를 사용하기 위해 AIX 어플리케이션을 사용자 정의하고자 하고

»

AIX에서 사용자 어플리케이션을 컴파일하고 있는 경우,

«

OS/400 PASE 프로그램을 컴파일하기 전에 하나 이상의 OS/400 헤더 파일이나 내보내기 파일을 사용자의 AIX 시스템에 복사해야 합니다.

사용자 프로그램과 OS/400 PASE의 호환성 분석

iSeries 서버에 대한 UNIX C 어플리케이션의 이식성을 산정하는 첫 번째 단계는 사용자 어플리케이션에서 사용되는 인터페이스를 분석하는 것입니다. 이 API 분석에서는 산업 표준이 아니고 OS/400에서 지원되지 않으면서 어플리케이션 내에서 사용되는 인터페이스를 식별합니다. 또한 표준과는 호환하지만 UNIX 기계와 비교하여 OS/400의 상이한 구조 때문에 지원 방식이 다른 인터페이스도 식별합니다.

API 분석 툴  은 front-end 및 back-end 프로세스로 구성됩니다. 프런트엔드 프로세스는 컴파일된 어플리케이션을 스캔하여 어플리케이션에서 사용하는 인터페이스(외부 기능 및 데이터)를 추출하고 이들 모든 인터페이스의 리스트를 생성합니다. 백엔드 프로세스는 이 인터페이스 목록을 입력으로 사용하여 일반 시스템 API 및 이의 지원 데이터베이스와 이 인터페이스를 비교합니다.

API 분석 툴의 프런트엔드 프로세스는 UNIX 쉘 스크립트입니다. 이는 `nm` 또는 `dump` 명령을 사용하여 어플리케이션의 외부 기호표에서 기호 정보를 찾아냅니다.

기호가 제거된 2진(binary)에는 툴이 분석을 수행할 수 있도록 충분한 동적 바인딩 정보가 들어 있습니다. 정적으로 바인드시킨 2진은 라이브러리 인터페이스를 분석에서 제거하지만 분석을 위해 시스템 호출 종속성은 보여줍니다.

컴파일 전 수행할 추가 분석

API 분석 툴을 통해 수집한 정보 외에 다음 정보도 수집해야 합니다.

- **어플리케이션에서 사용하는 라이브러리 리스트 확보:** 분석 툴은 사용자 어플리케이션이 사용하는 일부 표준 API에 대해 피드백을 제공하지만 다수의 공통 API 세트를 찾지는 않습니다. 라이브러리 분석은 사용자 어플리케이션이 사용하는 일부 미들웨어 API를 식별하는 데 도움이 됩니다. 각각의 사용자 명령과 공유 오브젝트에 대해 다음 명령을 실행하여 어플리케이션에 필요한 라이브러리 리스트를 얻을 수 있습니다.

  ```
  dump -H binary_name
  ```

- »

  사용자 코드에서 하드 코드 경로명 검사: 증명서를 변경하는 프로그램을 실행하거나 OS/400 PASE 환경 변수의 설정이 PASE_EXEC_QOPENSYS=N인 경우에도 사용자 프로그램이나 스크립트를 실행하려는 경우, 하드 코드 경로명을 변경해야 할 수 있습니다.

  /usr/bin/ksh는 (루트에서 시작하는) 절대 경로이므로 이를 찾을 수 없거나 바이트 스트림 파일이 아닐 경우 OS/400 PASE는 자동으로 /QOpenSys 파일 시스템에서 경로명 /QOpenSys/usr/bin/ksh를 탐색합니다. QShell 유틸리티 프로그램은 바이트 스트림 파일이 아니므로 OS/400 PASE는 원래(절대) 경로가 QShell 유틸리티 프로그램에 대한 기호 링크(예: /usr/bin/sh)인 경우에도 /QOpenSys 파일 시스템을 탐색합니다.

  «

AIX 소스 컴파일

사용자 프로그램이 AIX 인터페이스만을 사용하는 경우 필수 AIX 헤더를 사용하여 컴파일하고 AIX 라이브러리와 링크하여 OS/400 PASE를 위한 2진(binary)을 준비하십시오. OS/400 PASE는 AIX 제공 공유 라이브러리와 정적으로 바인드시킨 어플리케이션은 지원하지 않습니다.

OS/400 PASE 프로그램은 구조적으로 PowerPC용 AIX 프로그램과 일치합니다.

»

OS/400 PASE 옵션 33에는 컴파일러가 들어 있지 않습니다. AIX 시스템을 OS/400 PASE 프로그램을 컴파일하는데 사용하거나 OS/400 PASE 환경에서 프로그램을 컴파일하기 위해 OS/400 PASE에서 선택적으로 두 개의 AIX 컴파일러 제품 중 하나를 설치할 수 있습니다.

«

pSeries 서버에서 AIX 컴파일러 사용

AIX PowerPC용 ABI(Application Binary Interface)와 호환되는 출력을 생성하는 링크 프로그램과 AIX 컴파일러를 사용하여 OS/400 PASE 프로그램을 빌드할 수 있습니다. OS/400 PASE는 PowerPC에 존재하지 않는 POWER 구조 명령어(캐시 관리 POWER 명령어 제외)를 사용하는 2진에 명령어 에뮬레이션 지원을 제공합니다.

OS/400 PASE에서 AIX 컴파일러 사용

OS/400 PASE는 OS/400 PASE 환경에서 다음과 같이 단독적으로 사용가능한 AIX 컴파일러 중에서의 설치를 지원합니다.

- IBM VisualAge C++ Professional for AIX, 버전 6(5765-F56). (이 제품은 AIX 컴파일러용 IBM C를 포함합니다.)
- IBM C for AIX, 버전 6(5765-F57)

이 제품을 사용하면 iSeries 서버의 OS/400 PASE 환경 안에서 AIX 어플리케이션을 개발, 컴파일, 빌드 및 실행할 수 있습니다.

제품의 주문 및 설치에 대한 자세한 정보는 어플리케이션 팩토리 웹 사이트에서 OS/400 PASE에서 VisualAge C++ for AIX compiler 설치 페이지를 참조하십시오.

개발 툴

AIX에서 사용하는 대다수 개발 툴(예: ld, ar, make, yacc)이 OS/400 PASE에 들어 있습니다. 세부사항은 OS/400 PASE 쉘 및 유틸리티 주제를 참조하십시오. 다른 소스의 대다수 AIX 툴(예: 개발 소스 툴 gcc)도 OS/400 PASE에서 작동 가능합니다.

iSeries Tools for Developers PRPQ(5799-PTL)에도 iSeries 어플리케이션의 개발, 빌드 및 이식을 지원하기 위한 다양한 툴이 들어 있습니다. 이 PRPQ에 대한 자세한 정보는 Application Factory - iSeries Tools for Development 웹 사이트를 참조하십시오.

포인터 처리를 위한 컴파일러

- xlc 컴파일러는 -qlngdbl128 및 -qalign=natural의 조합을 사용하여 16바이트 정렬에 대해 제한된 지원을 제공합니다(유형 long double의 경우). 유형 ILEpointer에는 MI 포인터가 구조 내에서 16바이트로 정

렬되도록 보장하기 위해 이러한 컴파일러 옵션이 필요합니다. -qldbl128 옵션을 사용하면 유형 long double은 long double 필드에 대해 printf와 같은 조작을 처리할 수 있도록 libc128.a의 사용을 필요로 하는 128비트 유형으로 만들어집니다.

-qlngdbl128 옵션에 접근하여 libc128.a에 링크하는 손쉬운 방법은 xlc 명령 대신 xlc128 명령을 사용하는 것입니다.

- xlc/xlC 컴파일러는 현재 정적 변수나 자동 변수에 대해 16바이트 정렬을 강제하는 수단을 제공하지 않습니다. 다만 이 컴파일러는 구조 내에서 128비트 long double 필드에 대해 상대 정렬을 보장하는 정도입니다. malloc의 OS/400 PASE 버전은 16바이트 정렬 기억장치를 반드시 제공하므로 스택 기억장치의 16바이트 정렬이 가능합니다.
- 헤더 파일 as400_types.h는 유형 long long에서도 64비트 정수가 됩니다. xlc 컴파일러 옵션 -qlonglong이 이러한 구조를 보장해줍니다(단 xlc 컴파일러를 실행하는 모든 명령에 대해 디폴트는 아닙니다).

예

다음 예는 AIX 시스템에서 OS/400 PASE 프로그램을 컴파일하는 경우에 사용할 수 있도록 하기 위한 것입니다. 프로그램을 컴파일하기 위해 OS/400 PASE에 설치된 컴파일러를 사용하면, OS/400 고유 헤더 파일 또는 OS/400 고유 반출물의 위치에 대한 컴파일러 옵션을 지정할 필요는 없습니다. 그 이유는 OS/400 시스템의 디폴트 경로인 /usr/include/ 및 /usr/lib/에서 찾을 수 있기 때문입니다.

예 1: AIX 시스템의 다음 명령은 libc.a에서 내보내기 한 OS/400 고유 인터페이스를 사용할 수 있는 OS/400 PASE 프로그램 *testpgm*을 작성합니다.

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

이 예에서는 OS/400 고유 헤더 파일이 AIX 디렉토리 /usr/include에 복사되고 OS/400 고유 내보내기 파일이 AIX 디렉토리 /mydir에 복사되는 것으로 가정합니다.

예 2: 다음 예에서는 OS/400 고유 헤더와 내보내기 파일이 /pase/lib에 있는 것으로 가정합니다.

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

예 3: 다음 예는 예 2와 동일한 프로그램을 같은 옵션을 사용하여 빌드합니다. 그러나 컴파일된 어플리케이션이 스레드세이프 런타임 라이브러리에 링크되도록 보장하기 위해 xlc_r 명령이 멀티스레드 프로그램에 사용됩니다.

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

예에서 iSeries용 DB2 UDB 호출 레벨 인터페이스(CLI)에 대해 OS/400 PASE 지원을 사용하는 경우, 빌드 명령에 -bI:/pase/include/libdb400.exp도 지정해야 합니다.

-bI 지시문은 ld 명령에 매개변수를 전달하도록 컴파일러에 지시합니다. 지시문은 라이브러리에서 내보낸 기호가 들어 있는 내보내기 파일을 어플리케이션이 가져오도록 지정합니다.

OS/400 PASE 프로그램을 iSeries 서버에 복사

OS/400 PASE에서 실행하려는 AIX 2진(binary)을 통합 파일 시스템에 복사하십시오. 통합 파일 시스템에서 사용할 수 있는 파일 시스템은 모두 OS/400 PASE 내에서 사용할 수 있습니다. 통합 파일 시스템에 대한 자세한 정보는 통합 파일 시스템 주제를 참조하십시오.

플랫폼 사이에서 파일을 이동하는 경우 문제를 일으킬 수도 있는 다음과 같은 차이점에 유의하십시오.

- 어플리케이션이 대소문자를 구분하는 경우 어플리케이션을 /QOpenSys 파일 시스템으로 이동하거나 대소문자 구분으로 작성된 사용자 정의 파일 시스템으로 이동하십시오.
- AIX 및 OS/400은 텍스트 파일(예: 파일 및 쉘 스크립트)에서 서로 다른 행 종료 문자를 사용합니다.

파일 전송

다음 중 한 방법으로 OS/400 PASE 프로그램 및 관련 파일을 iSeries 서버와의 사이에서 전송 및 수신할 수 있습니다.

- 파일 전송 프로토콜(FTP)
- SMB(Server Message Block)(16 페이지 참조)
- 리모트 파일 시스템(17 페이지 참조)

파일 전송 프로토콜(FTP)을 사용하여 프로그램 복사

OS/400 FTP 디먼과 클라이언트를 사용하여 OS/400 통합 파일 시스템과 파일을 전송 및 수신할 수 있습니다. 2진 코드로 파일을 전송하십시오. 이 모드를 설정하려면 FTP 부속 명령 binary를 사용하십시오.

통합 파일 시스템에 파일을 배치할 때에는 명명 형식 1(OS/400 FTP 명령의 NAMEFMT 1 부속 명령)을 사용해야 합니다. 이 형식을 사용할 경우 UNIX 경로명을 사용할 수 있고 이 형식은 스트림 파일에 파일을 전송합니다. 명명 형식 1로 진입하려면 다음 중 하나를 수행하십시오.

- UNIX 경로명을 사용하여 디렉토리를 변경하십시오. 그러면 세션이 자동으로 이름 형식 1에 놓입니다. 이 방법을 사용할 경우 첫 번째 디렉토리 앞에 슬래시(/)가 붙습니다. 예를 들면 다음과 같습니다.

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

- 리모트 클라이언트에 대해 FTP 부속 명령 quote site namefmt 1을 사용하거나 namefmt 1을 로컬 클라이언트로 사용하십시오.

FTP에 대한 자세한 정보는 FTP 주제를 참조하십시오.

SMB(Server Message Block)를 사용하여 프로그램 복사

OS/400은 SMB 클라이언트와 서버 구성요소를 지원합니다. NetServer가 구성되어 실행 중이면 OS/400 PASE는 /QNTC 파일 시스템을 통해 네트워크의 SMB 서버에 대한 액세스권을 갖습니다. UNIX 플랫폼에서는 SAMBA 서버가 같은 서비스를 제공해야 합니다. 구성되어 작동 가능한 UNIX 시스템(AIX 등)을 설치하면 OS/400 PASE에서 디렉토리와 파일을 사용할 수 있게 만들 수 있습니다.

리모트 파일 시스템을 사용하여 프로그램 복사

OS/400에서는 NFS(Network File System) 파일 시스템을 통합 파일 시스템 파일 공간의 마운트 위치에 마운트할 수 있습니다. AIX는(DFS 대 NFS 및 AFS 대 NFS 변환 프로그램을 사용하여) NFS, DFS(Distributed File System)(TM) 및 AFS(Andrew File System)(R)를 지원하므로 OS/400은 이들 파일 시스템을 내보내기 및 마운트할 수 있습니다. 즉, OS/400 PASE 어플리케이션이 이들 파일 시스템을 사용할 수 있습니다. 디렉토리 경로 또는 액세스 중인 파일에 대한 OS/400 사용자 프로파일의 사용자 ID 번호와 그룹 ID 번호를 통해 보안 권한이 검증됩니다. 여러 플랫폼에서 같은 사람으로 인식되는 사용자 프로파일은 모든 시스템에서 같은 사용자 ID를 가져야 합니다.

OS/400은 NFS 서버로 사용하는 것이 가장 적합합니다. 이 경우 사용자의 AIX 시스템에서 OS/400 통합 파일 시스템의 디렉토리로 마운트하게 되며 프로그램 빌드 시 AIX는 OS/400에 직접 프로그램을 기록합니다.

주: OS/400 NFS는 현재 멀티스레드 어플리케이션에서 지원되지 않습니다.

대소문자 구분

UNIX 시스템 인터페이스는 보통 대문자와 소문자를 구분합니다. OS/400에서는 항상 대소문자가 구분되는 것은 아닙니다. 대소문자 구분이 기존 코드를 복잡하게 만드는 경우 등 몇몇 상황에 특히 유의해야 합니다.

디렉토리나 파일 기초에서의 대소문자 구분은 OS/400에서 사용 중인 파일 시스템에 따라 달라집니다. /QOpenSys 파일 시스템은 대소문자를 구분하므로 대소문자가 구분되는 사용자 정의 파일 시스템(UDF)을 작성할 수 있습니다. 여러 가지 파일 시스템에 대한 정보는 통합 파일 시스템 주제(및 특히 통합 파일 시스템의 파일 시스템: 비교 주제)를 참조하십시오.

또한 OS/400의 사용자 ID와 그룹 ID는 반드시 대문자로 리턴된다는 점에도 유의해야 합니다.

예

다음은 대소문자 구분으로 인해 발생할 수 있는 몇 가지 문제점입니다.

예 1: 이 예에서 쉘은 `readdir()`이 리턴하는 것에 대한 총칭명 접두부의 문자 비교입니다. 그러나 QSYS.LIB 파일 시스템은 대문자로 된 디렉토리 항목을 리턴하므로 어떤 항목도 소문자의 총칭명 접두부와 일치하지 않습니다.

```
$ 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
```

예 2: 이 예는 첫 번째 예와 유사하지만 이 예에서는 쉘이 아니라 `find` 유틸리티가 비교를 수행합니다.

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

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

예 3: ps 유틸리티는 대소문자가 구분되는 사용자명을 기대하므로 -u 옵션에 대해 지정되는 대문자 이름과 OS/400 PASE 런타임 함수 getpwuid()가 리턴하는 소문자 이름 사이의 일치를 인식하지 못합니다.

```
$ 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
```

통합 파일 시스템 파일의 행 종료 문자

OS/400 PASE 프로그램의 소스인 AIX 어플리케이션에서는 (파일 및 쉘 스크립트에 있는) 행이 라인 피드(LF)로 끝나야 합니다. 그러나 PC 소프트웨어 및 일반적인 OS/400 소프트웨어는 캐리지 리턴과 라인 피드(CRLF)를 사용하여 행을 끝내는 경우가 종종 있습니다.

FTP에 사용된 CRLF

이러한 차이가 문제를 초래할 수 있는 한 예가 FTP를 사용하여 소스 파일과 쉘 스크립트를 AIX에서 iSeries로 전송하는 경우입니다. 표준 FTP는 텍스트 모드에서 송신된 자료를 호출하여 행의 끝에서 캐리지 리턴과 라인 피드(CRLF)를 사용합니다. AIX에서 FTP 유틸리티는 텍스트 모드에서 인바운드 파일을 처리할 때 캐리지 리턴(CR)을 제거합니다. OS/400 FTP는 항상 자료 스트림에 제시된 내용을 정확히 기록하고 텍스트 모드에 대해 CRLF를 반드시 보유하는데, 이로 인해 OS/400 PASE 런타임과 유틸리티에서 문제가 초래됩니다.

가능하면 UNIX 시스템의 2진 모드 전송을 사용하여 이 문제를 방지하십시오. 대부분의 경우 퍼스널 컴퓨터에서 전송된 텍스트 파일은 CRLF 구분 행이 있습니다. 먼저 AIX에 파일을 전송하면 문제가 정정됩니다. 다음 조치는 현재 디렉토리에 있는 파일에서 CR을 제거하기 위한 수단으로 제공된 것입니다.

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

iSeries 및 PC 편집기에 사용된 CRLF

또한 iSeries 서버에 있는 편집기나 사용자 워크스테이션의 편집기(예: Windows(R) Notepad 편집기)를 사용하여 파일이나 쉘 스크립트를 편집할 때에도 문제가 발생할 수 있습니다. 이들 편집기는 CRLF를 새 행 분리자로 사용하지 OS/400 PASE가 기대하는 LF로 사용하지 않습니다.

CRLF를 새 행 분리자로 사용하지 않는 여러 편집기(예: ez 편집기)를 사용할 수 있습니다.

IBM PartnerWorld 웹 사이트의 여러 iSeries tools for developers 리스트를 참조하십시오.

OS/400 기능을 사용하도록 OS/400 PASE 프로그램 사용자 정의

AIX 어플리케이션이 시스템 제공 OS/400 PASE 공유 라이브러리가 직접 지원하지 않는 OS/400 기능을 활용하도록 하려면 어플리케이션을 준비하기 위한 약간의 추가 단계를 수행해야 합니다.

먼저 OS/400 고유 기능에 대한 사용자 액세스를 조정하는 필수 OS/400 PASE 런타임 함수를 호출하도록 AIX 어플리케이션을 사용자 정의해야 합니다.

둘째

»

AIX 시스템에서 OS/400 PASE 프로그램을 컴파일하는 중일 경우 사용자 정의된 어플리케이션을 컴파일하기 전에 다음 단계를 수행해야 합니다.

«

- AIX 시스템에 필수 OS/400 고유 헤더 파일 복사
- AIX 시스템에 필수 OS/400 고유 내보내기 파일 복사

OS/400 PASE를 OS/400 기능과 통합하는 데 대한 자세한 정보는 다음 주제를 참조하십시오.

- OS/400 PASE 프로그램에서 OS/400 프로그램과 기능 호출
- OS/400 PASE 프로그램과 OS/400 기능의 대화 방식

헤더 파일 복사

OS/400 PASE는 OS/400 PASE 고유 지원을 위한 헤더 파일을 가지고 있어서 표준 AIX 런타임을 보강합니다. OS/400 PASE(19 페이지 참조) 및 OS/400 오퍼레이팅 시스템(20 페이지 참조)으로 이러한 보강이 제공됩니다. iSeries 서버의 헤더 파일을 헤더 파일 탐색 경로에 있는 AIX 기계에 복사하십시오.

다음의 AIX 디렉토리나 사용 중인 컴파일러에 대한 헤더 파일 탐색 경로에 있는 디렉토리에 이 파일을 복사하십시오.

```
/usr/include
```

/usr/include 이외의 디렉토리를 사용하는 경우 AIX 컴파일러 명령에서 -I 옵션을 사용하여 헤더 파일 탐색 경로에 헤더 파일을 추가할 수 있습니다.

파일 복사에 대한 자세한 정보는 OS/400 PASE 프로그램을 iSeries 서버에 복사를 참조하십시오.

OS/400 PASE 헤더 파일 복사

OS/400 PASE 헤더 파일은 다음의 OS/400 디렉토리에 있습니다.

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

OS/400 PASE는 다음의 헤더 파일을 제공합니다.

as400_protos.h	OS/400 PASE 대 ILE. 기타 OS/400 고유 함수를 제공합니다.
as400_types.h	ILE 호출을 위한 고유한 OS/400 매개변수 유형 이 헤더 파일은 16바이트 기계 인터페이스(MI) 포인터에 대해 유형 ILEpointer를 선언합니다. 이 유형은 유형 long double에서도 128비트 필드로 만들어줍니다. as400_types.h에 선언된 기타 유형은 유형 long long에서 64비트 정수가 됩니다. as400_types.h에서 적절한 크기와 정렬 유형이 선언되도록 보장하기 위해 옵션 -qlngdbl128, -qalign=natural 및 -qlonglong를 사용하여 AIX 컴파일러를 실행해야 합니다.
os400msg.h	OS/400 메세지를 송수신하기 위한 함수

OS/400 헤더 파일 복사

OS/400 제공 헤더 파일은 다음 디렉토리에 있습니다.

```
/QIBM/include
```

사용자 어플리케이션에 OS/400 API 헤더 파일이 필요한 경우 먼저 이 파일을 EBCDIC에서 ASCII로 변환한 다음 변환된 파일을 AIX 디렉토리에 복사하십시오.

EBCDIC 텍스트 파일을 ASCII로 변환하는 한 방법으로 OS/400 PASE Rfile 유틸리티를 사용할 수 있습니다.

다음 예에서는 OS/400 PASE Rfile 유틸리티를 사용하여 OS/400 헤더 파일 /QIBM/include/qusec.h를 읽고 자료를 OS/400 PASE CCSID로 변환하며 각 행의 끝에 추가된 공백을 제거하며 결과를 바이트 스트림 파일 ascii_qusec.h에 기록합니다.

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

내보내기 파일 복사

iSeries 서버에서 AIX 디렉토리로 내보내기 파일을 복사하십시오.

다음의 OS/400 디렉토리에 있는 내보내기 파일은 OS/400 고유 함수에 대한 액세스를 필요로 하는 어플리케이션을 빌드하는 좋은 방법입니다.

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

이 파일을 임의의 AIX 디렉토리에 복사할 수 있습니다. ld 명령(또는 컴파일러 명령)의 -bI: 옵션을 사용하여 AIX 시스템의 공유 라이브러리에 없는 기호를 정의하십시오.

OS/400 PASE는 다음의 내보내기 파일을 제공합니다.

as400_libc.exp	libc.a에 있는 OS/400 고유 함수에 대한 내보내기 파일 as400_libc.exp 파일은 이들 라이브러리의 AIX 버전에서 내보내지 않은 OS/400 PASE libc.a 버전으로부터의 모든 내보내기를 정의합니다.

libdb400.exp	OS/400 데이터베이스 함수에 대한 내보내기 파일 libdb400.exp 파일은 OS/400 PASE libdb400.a 라이브러리로부터의 내보내기를 정의합니다(iSeries용 DB2 UDB 호출 레벨 인터페이스 지원).

파일 복사에 대한 자세한 정보는 OS/400 PASE 프로그램을 iSeries 서버에 복사를 참조하십시오.

OS/400 기능에 액세스하기 위한 OS/400 PASE API

OS/400 PASE는 ILE 코드 및 기타 OS/400 기능에 액세스하기 위한 다수의 API를 제공합니다. 컴파일러가 수행하는 것과 비교하여 사용자 스스로 준비와 구조 빌드를 수행하려는 빈도에 따라 사용할 API를 선택할 수 있습니다. 자세한 내용은 OS/400 PASE API를 참조하십시오.

OS/400 환경에서 OS/400 PASE 프로그램 사용

OS/400 PASE 프로그램은 사용자의 작업에서 실행 중인 다른 OS/400 프로그램을 호출할 수 있고 다른 OS/400 프로그램은 사용자의 OS/400 PASE 프로그램에 있는 프로시듀어를 호출할 수 있습니다. OS/400 PASE 프로그램을 사용자의 컴퓨팅 환경에 통합하는 방법에 대해서는 다음 주제를 참조하십시오.

OS/400 프로그램에서 OS/400 PASE 프로그램 실행
작업에서 OS/400 PASE 프로그램을 시작하고 ILE 프로그램에서 OS/PASE 프로시듀어를 호출하는 데 대한 정보와 예를 제공합니다.

OS/400 PASE 프로그램에서 OS/400 프로그램과 기능 호출
OS/400 PASE 프로그램에서 ILE 프로시듀어, OS/400 프로그램 및 CL 명령을 호출하는 데 대한 정보와 예를 제공합니다.

OS/400 PASE 프로그램과 OS/400 기능의 대화 방식
OS/400 PASE 프로그램의 사용 및 OS/400 기능과의 대화 방식에 대한 정보를 제공합니다.

OS/400 PASE 프로그램 및 프로시듀어 실행

OS/400 PASE는 사용자의 OS/400 PASE 프로그램을 실행하기 위한 여러 가지 방법을 제공합니다.

»

QP2SHELL()및 QP2SHELL2()

«

자신을 호출한 작업에서 OS/400 PASE 프로그램을 실행하는 OS/400 프로그램.

QP2TERM()
대화식 쉘 환경에서 OS/400 PASE 프로그램을 실행하는 OS/400 프로그램.

Qp2RunPase()
OS/400 PASE 프로그램을 시작 및 실행하기 위해 ILE 프로시듀어 내에서 호출하는 ILE 프로시듀어.

Qp2CallPase()
OS/400 PASE 환경이 이미 실행 중인 작업에서 OS/400 PASE 프로그램을 실행하기 위해 ILE 프로시듀어 내에서 호출하는 ILE 프로시듀어.

환경 변수에 대한 작업에서는 OS/400 PASE 환경이 OS/400 환경과 대화하는 방식에 대해 설명합니다.

OS/400 PASE 프로그램에 대해 작업할 수 있는 ILE 프로시듀어

OS/400 PASE는 사용자의 ILE 코드가(OS/400 PASE 프로그램에서의 특별한 프로그래밍 없이도) OS/400 PASE 서비스에 액세스할 수 있도록 하는 다수의 ILE 프로시듀어 API를 제공합니다.

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

자세한 내용은 OS/400 PASE ILE 프로시듀어 API를 참조하십시오.

ILE 스레드에 접속

OS/400 PASE가 작성하지 않은 스레드(예: Java 스레드 또는 ILE pthread_create가 작성한 스레드)에서 실행되는 ILE 코드에서 OS/400 PASE 프로그램의 프로시듀어를 호출할 수 있습니다. Qp2CallPase는 자동으로 ILE 스레드를 (상응하는 OS/400 PASE pthread 구조를 작성하는) OS/400 PASE에 접속합니다. 단 OS/400 PASE 프로그램이 시작될 때 OS/400 PASE 환경 변수 PASE_THREAD_ATTACH가 Y로 설정된 경우에 한합니다.

OS/400 PASE에서 OS/400 프로그램으로 결과 리턴

OS/400 _RETURN() 함수를 사용하여 OS/400 PASE 프로그램을 호출하고 OS/400 PASE 환경을 편집하지 않고도 결과를 리턴할 수 있습니다. 이로써 OS/400 PASE 프로그램을 시작한 다음 QP2SHELL2(QP2SHELL이 아님) 또는 Qp2RunPase API 이후 해당 프로그램에서 프로시듀어를 호출할 수 있습니다.

QP2SHELL()을 사용하여 OS/400 PASE 프로그램 실행

임의의 OS/400 명령행과 고급 언어 프로그램, 일괄처리 작업 또는 대화식 작업에서 OS/400 PASE 프로그램을 실행하려면 OS/400 PASE 쉘 프로그램(QP2SHELL 또는

QP2SHELL2)

실행을 사용하십시오. 이들 프로그램은 자신을 호출한 작업에서 OS/400 PASE 프로그램을 실행합니다. OS/400 PASE 프로그램의 이름이 프로그램에서 매개변수로 전달됩니다. 이 프로그램의 사용 방법에 대한 자세한 내용은 QP2SHELL() 및 QP2SHELL2() 설명을 참조하십시오.

QP2SHELL() 프로그램은 OS/400 PASE 프로그램을 새로운 활성 그룹에서 실행합니다.

QP2SHELL2() 프로그램은 호출자의 활성 그룹에서 실행됩니다.

다음 예는 OS/400 명령행에서 ls 명령을 실행합니다.

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

변수를 사용하여 QP2SHELL()에 값 전달

CL 변수를 사용하여 QP2SHELL()에 값을 전달하는 경우 반드시 변수를 널(null) 종료해야 합니다. 예를 들어 위의 예를 다음과 같은 방식으로 코딩해야 합니다.

```
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
```

QP2TERM()을 사용하여 OS/400 PASE 프로그램 실행

QP2TERM() 프로그램에서 OS/400 PASE 대화식 단말기 세션을 시작하십시오. 다음 명령은 디폴트 Korn 쉘 프롬프트 (/QOpenSys/usr/bin/sh)를 세션에 기록합니다.

```
call qp2term
```

이 프롬프트에서 별도의 일괄처리 작업으로 OS/400 PASE 프로그램을 실행하십시오. QP2TERM()은 대화식 작업을 사용하여 출력을 표시하고 파일 stdin, stdout 및 stderr에 대한 입력을 일괄처리 작업으로 승인합니다.

Korn 쉘은 디폴트이지만 프로그램에 전달하려는 인수 스트링뿐 아니라 실행하려는 OS/400 PASE 프로그램의 경로명을 선택적으로 지정할 수 있습니다.

QP2TERM()을 시작한 대화식 세션에서 임의의 OS/400 PASE 프로그램과 임의의 유틸리티를 실행할 수 있습니다. stdout 및 stderr은 단말기 화면에 기록되고 화면이동됩니다.

OS/400 프로그램 내에서 OS/400 PASE 프로그램 실행

Qp2RunPase() API를 사용하여 OS/400 PASE 프로그램을 실행하십시오. 프로그램명, 인수 스트링 및 환경변수를 지정하십시오. ILE 프로그램에서 이를 사용하는 방법에 대한 자세한 정보는 Qp2RunPase() API 설명을 참조하십시오.

Qp2RunPase() API는 자신이 호출된 작업에서 OS/400 PASE 프로그램을 실행합니다. 이 API는(필요한 공유 라이브러리를 포함하여) OS/400 PASE 프로그램을 로드한 후 프로그램에 제어를 전송합니다.

이 API는 QP2SHELL() 및 QP2TERM()보다 OS/400 PASE의 실행 방식에 대해 더 많은 제어를 부여합니다.

ILE 프로그램에서 이 API를 사용하는 방법에 대한 예를 보려면 프로그램 예를 참조하십시오.

예: OS/400 프로그램 내에서 OS/400 PASE 프로그램 실행: 다음의 ILE 프로그램(면책사항 참조)은 OS/400 PASE 프로그램을 호출합니다. 이 예의 다음에는 이 프로그램이 호출하는 OS/400 PASE 코드의 예가 나와 있습니다.

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

/* QP2RunPase()에 대한 포함 파일. */

#include <qp2user.h>

/*****************************************
  샘플:
  QP2RunPase()를 사용하여 하나의 스트링
  매개변수를 전달하여 OS/400 PASE 프로그램을
  호출하는 간단한 ILE C 프로그램.
  컴파일 예:
    CRTCMOD MODULE(MYLIB/SAMPLEILE) SRCFILE(MYLIB/QCSRC)
```

```
    CRTPGM PGM(MYLIB/SAMPLEILE)
*****************************************/

void main(int argc, char*argv[])
 {
  /* PASE 프로그램의 경로명 */
  char *PasePath = "/home/samplePASE";
  /* QP2RunPase()의 리턴 코드 */
  int rc;
  /* OS/400 PASE 프로그램에 전달되는
     매개변수 */
  char *PASE_parm = "My Parm";
  /* OS/400 PASE 프로그램에 대한 인수 리스트로
     포인터 리스트에 대한 포인터입니다. */
  char **arg_list;
  /* 인수 리스트를 할당하십시오. */
  arg_list =(char**)malloc(3 * sizeof(*arg_list));
  /* 프로그램명을 첫 번째 요소로 설정하십시오. 이것은 UNIX 규약입니다. */
  arg_list[0] = PasePath;
  /* 매개변수를 첫 번째 요소로 설정하십시오. */
  arg_list[1] = PASE_parm;
  /* 인수 리스트의 마지막 요소는 반드시 널(null)이어야 합니다. */
  arg_list[2] = 0;
  /* OS/400 PASE 프로그램을 호출하십시오. */
   rc = Qp2RunPase(PasePath, /* 경로명 */
      NULL,           /* ILE를 호출하기 위한 기호. 이 샘플에서는 사용되지 않음 */
      NULL,           /* ILE 호출에 대한 기호 자료. 여기서는 사용되지 않음 */
      0,              /* ILE 호출에 대한 기호 자료. 여기서는 사용되지 않음 */
      819,            /* OS/400 PASE에 대한 ASCII CCSID */
      arg_list,       /* OS/400 PASE 프로그램에 대한 인수 */
      NULL);          /* 환경 변수 리스트. 이 샘플에서는 사용되지 않음 */
 }
```

다음의 OS/400 PASE 프로그램(면책사항 참조)은 위의 ILE 프로그램에서 호출한 것입니다.

```
#include <stdio.h>

/*****************************************
  샘플:
  QP2RunPase()를 사용하고 하나의 스트링
  매개변수를 채택하여 ILE에서 호출한
  간단한 OS/400 PASE 프로그램.
  ILE 샘플 프로그램은 이 프로그램이
  /home/samplePASE에 있는 것으로 압니다.
  AIX에서 컴파일한 후 OS/400에 ftp 전송하십시오.
  ftp 전송을 하려면 다음 명령을 사용하십시오.
  > binary
  > site namefmt 1
  > put samplePASE /home/samplePASE
*****************************************/

int main (int argc, char *argv[])
{
    /* 전달된 인사와 매개변수를 인쇄하십시오. argv[0]가 프로그램 이름이므로
       argv[1]은 매개변수입니다. */
     printf("Hello from OS/400 PASE program %s. Parameter value is \"%s\".\n", argv[0], argv[1]);

  return 0;
}
```

OS/400 PASE 프로그램 내에서 OS/400 PASE 프로시듀어 호출

먼저 Qp2RunPase() API가 시작되어 작업에서 OS/400 PASE 프로그램을 실행합니다. OS/400 PASE가 이미 해당 작업에서 사용 중일 경우 오류가 리턴됩니다.

이미 OS/400 PASE 프로그램을 실행 중인 작업에서 OS/400 PASE 프로시듀어를 호출하려면 Qp2CallPase() and Qp2CallPase2() API를 사용하십시오.

Qp2CallPase() API의 사용 방법에 대한 예를 보려면 예: OS/400 프로그램 내에서 OS/400 PASE 프로시듀어 호출을 참조하십시오.

예: OS/400 프로그램 내에서 OS/400 PASE 프로시듀어 호출: 다음의 ILE 프로그램(면책사항 참조)은 OS/400 PASE 프로시듀어를 호출합니다.

```
#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;

    /*
     * QP2SHELL2를 호출하여 OS/400 PASE 프로그램
     * /usr/lib/start32를 실행하십시오. 이는 OS/400 PASE를
     * 32비트 모드로 시작합니다(리턴 시 활동 상태를 유지합니다).
     */
    QP2SHELL2("/usr/lib/start32");

    /*
     * Qp2dlopen은 첫 번째 인수가 널(null) 포인터일
     * 경우 글로벌명 공간을 엽니다(새 공유 실행 파일을
     * 로드하지 않음). Qp2dlsym은 OS/400 PASE getpid
     * 서브루틴(공유 라이브러리 libc.a에서 내보냄)을
     * 찾습니다.
     */
    id = Qp2dlopen(NULL, QP2_RTLD_NOW, JOB_CCSID);
    getpid_pase = Qp2dlsym(id, "getpid", JOB_CCSID, NULL);

    /*
     * Qp2CallPase를 호출하여 OS/400 PASE getpid
     * 함수를 실행하고 결과를 인쇄하십시오. 함수 결과가
     * -1인 경우 Qp2errnop를 사용하여 OS/400 PASE errno를
     * 찾아 인쇄하십시오.
     */
    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());

    /*
     *  Qp2dlopen 인스턴스를 닫고 Qp2EndPase를
     *  호출하여 이 작업에서 OS/400 PASE를 종료하십시오.
     */
    Qp2dlclose(id);
    Qp2EndPase();
  return 0;
}
```

환경 변수에 대한 작업

OS/400 PASE 환경 변수는 ILE 환경 변수와 독립적입니다. 한 환경에서 변수를 설정하여도 다른 환경에 영향이 미치지 않습니다. 그러나 OS/400 PASE 프로그램 실행 방법에 따라 ILE에서 OS/400 PASE로 변수를 복사할 수 있습니다.

대화식 OS/400 PASE 세션의 환경 변수

QP2SHELL() 및 QP2TERM()을 사용하여 OS/400 PASE를 시작한 경우에만 ILE 환경 변수가 OS/400 PASE로 전달됩니다. 환경 변수에 대한 작업(WRKENVVAR) 명령을 사용하여 OS/400 PASE를 시작하기 전에 필요한 대로 환경 변수를 변경, 추가 또는 삭제할 수 있습니다.

호출된 OS/400 PASE 세션의 환경 변수

OS/400 PASE가 (Qp2RunPase() API를 사용하여) 프로그램 호출에서 시작된 경우 환경 변수에 대한 완전한 제어를 가지십시오. OS/400 PASE 프로그램을 호출한 ILE 환경과 아무런 관계가 없는 환경 변수를 전달할 수 있습니다.

»

CL 명령을 실행하기 전에 ILE에 환경 변수 복사

systemCL 런타임 함수에서 옵션을 사용하여 CL 명령을 실행하기 전에 ILE 환경에 OS/400 PASE 환경 변수를 복사할 수 있습니다. 이것은 OS/400 PASE system 유틸리티의 디폴트 작동이기도 합니다.

«

자세한 정보는 OS/400 PASE 환경 변수 주제를 참조하십시오.

OS/400 PASE 프로그램에서 OS/400 프로그램과 프로시듀어 호출

OS/400 PASE는 OS/400 기능에 대한 통합된 액세스를 제공하는 ILE 프로시듀어, Java 프로그램, OPM 프로그램, OS/400 API 및 CL 명령을 호출하기 위한 메소드를 제공합니다.

다음 주제에서는 OS/400 PASE 환경에서 호출할 경우의 지침과 예를 제공합니다.

ILE 프로시듀어 호출
OS/400 PASE에서 ILE 프로시듀어를 호출할 수 있으려면, OS/400 PASE 프로그램에서 발생하는 호출을 처리하도록 ILE 프로시듀어를 설정해야 합니다. 또한 컴파일된 AIX 프로그램에서 프로그램 변수와 구조를 설정해야 합니다.

OS/400 프로그램 호출
OS/400 PASE 프로그램 내에서 OS/400 프로그램을 호출할 수 있습니다.

OS/400 명령 실행
OS/400 PASE 프로그램 내에서 CL 명령을 실행할 수 있습니다.

OS/400 프로그램과 프로시듀어에 대한 일반 구성 요구사항

OS/400 PASE 프로그램 환경에서 OS/400 환경으로 호출할 경우 다음과 같은 이유로 활성 그룹에 대해 반드시 *CALLER를 사용하여 OS/400 프로그램을 컴파일해야 합니다.

- 활성 그룹에서 실행되며 OS/400 PASE(Qp2RunPase API로 호출됨)를 시작한 코드만이 ILE API(예: Qp2CallPase)를 사용하여 OS/400 PASE 프로그램과 대화할 수 있습니다.
- ILE 런타임은 멀티스레드 작업 중인 활성 그룹을 파기해야 할 경우 전체 작업(및 OS/400 PASE)을 종료할 수 있습니다(또한 OS/400 PASE `fork`에서 시작한 모든 작업은 멀티스레드가 가능합니다). ACTGRP(*CALLER)를 사용하여 사용자가 원하지 않을 때 작업이 종료하는 것을 방지할 수 있습니다.

`systemCL` 런타임 함수를 사용하여 CL 명령(CALL 명령 등)을 멀티스레드가 가능하지 않은 별도의 작업으로 실행하도록 하여 멀티스레드 가능 작업으로 실행되는 문제점을 방지할 수 있습니다.

ILE 프로시듀어 호출

OS/400 PASE 프로그램에서 ILE 프로시듀어를 호출하려면 사용자 코드에 다음의 API 호출을 작성하십시오.

1. OS/400 PASE를 시작한 프로시듀어와 연관된 ILE 활성 그룹에 바인드 프로그램을 로드하십시오. 이 작업을 수행하려면 _ILELOAD() API를 사용하십시오.

   OS/400 PASE를 시작한 활성 그룹에서 바인드 프로그램이 이미 사용 중이라면 이 단계가 필요하지 않습니다. 이 경우 _ILESYM 단계로 진행하여 활성화 마크 매개변수에 0의 값을 사용하여 현재 활성 그룹에

있는 모든 활성 바인드 프로그램에서 모든 기호를 탐색할 수 있습니다.

2. ILE 바인드 프로그램의 활성화에서 내보낸 기호를 찾아 16바이트 태그 포인터를 기호에 대한 데이터나 프로시듀어로 리턴하십시오. 이 작업을 수행하려면 _ILESYM() API를 사용하십시오.
3. OS/400 PASE 프로그램에서 ILE 프로시듀어로 제어를 전송하기 위한 ILE 프로시듀어를 호출하십시오. 이 작업을 수행하려면 _ILECALL() 또는 _ILECALLX() API를 사용하십시오.

또한 OS/400 PASE 프로그램에서 ILE 프로시듀어를 호출할 때에는 다음 타스크도 수행해야 합니다.

- teraspace에 대해 ILE 프로시듀어 작동
- 텍스트를 적절한 CCSID(코드화 문자 세트 ID)로 변환
- 변수 및 구조 설정

teraspace에 대해 ILE 프로시듀어 작동

OS/400 PASE에서 호출하는 모든 ILE 모듈은 teraspace 옵션 세트를 *YES로 설정하여 컴파일해야 합니다. ILE 모듈을 이와 같이 컴파일하지 않을 경우 OS/400 PASE 어플리케이션에 대한 작업 로그에 MCH4433 오류 메세지(목표 프로그램 &2에 대해 유효하지 않은 기억장치 모델)가 수신됩니다. 자세한 정보는 ILE 개념  책을 참조하십시오.

텍스트를 적절한 CCSID(코드화 문자 세트 ID)로 변환

ILE와 OS/400 PASE 사이에서 전달되는 텍스트는 전달 이전에 적절한 CCSID로 변환해야 합니다. 이러한 변환을 수행하지 않을 경우 문자 변수에 예측하지 못한 값이 포함될 수 있습니다.

변수 및 구조 설정

OS/400 PASE 프로그램에서 ILE를 호출하는 경우 변수와 구조를 설정해야 합니다. 필수 헤더 파일이 AIX 시스템에 복사되었는지 확인해야 하고 서명, 결과 유형 및 인수 리스트 변수를 설정해야 합니다.

- **헤더 파일:** OS/400 PASE 프로그램에는 ILE를 호출하기 위한 헤더 파일 as400_types.h 및 as400_protos.h가 들어 있어야 합니다. as400_type.h 헤더 파일에는 OS/400 고유 인터페이스에 사용되는 유형에 대한 정의가 들어 있습니다.
- **서명:** 서명 구조에는 OS/400 PASE와 ILE 사이에서 전달되는 인수의 순서와 유형에 대한 설명이 들어 있습니다. 호출 중인 ILE 프로시듀어에서 관리하는 유형에 대한 코드화는 as400_types.h 헤더 파일에서 찾을 수 있습니다. 서명에 4바이트 미만의 고정 소수점 인수 또는 8바이트 미만의 부동 소수점 인수가 들어 있는 경우 다음의 pragma 인수를 사용하여 ILE C 코드를 컴파일해야 합니다.

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

  이 pragma 인수가 없으면 ILE에 대한 표준 C 연계에서 1바이트 및 2바이트 정수 인수를 4바이트로 확장하고 4바이트 부동 소수점 인수를 8바이트로 확장해야 합니다.
- **결과 유형:** 결과 유형은 간단하며 리턴 유형 C와 유사하게 작동합니다.

- **인수 리스트:**

  인수 리스트는 서명 배열의 항목에서 지정한 유형의 정확한 필드 순서를 갖는 구조여야 합니다.

  size_ILEarglist() 및 build_ILEarglist() API를 사용하여 서명에 기반한 인수 리스트를 동적으로 빌드할 수 있습니다.

OS/400 PASE에서 ILE 프로시듀어를 호출하는 프로세스의 예를 보려면 예: ILE 프로시듀어 호출을 참조하십시오.

예: ILE 프로시듀어 호출: 다음의 코드 예(면책사항 참조)는 서비스 프로그램의 일부인 ILE 프로시듀어(35 페이지 참조)를 호출하는 OS/400 PASE 코드(30 페이지 참조)와 프로그램을 작성하는 컴파일러 명령(37 페이지 참조)을 보여줍니다. 이 예에는 두 개의 UNIX 프로시듀어가 있습니다. 각 프로시듀어는 ILE 프로시듀어에 대한 여러 가지 작업 방식을 보여주지만 두 프로시듀어 모두 동일한 ILE 프로시듀어를 호출합니다. 첫 번째 프로시듀어는 OS/400 PASE 제공 방법을 사용한 _ILECALL API에 대한 자료 구조 빌드를 보여줍니다. 그런 후 두 번째 프로시듀어는 인수 리스트를 수동으로 빌드합니다.

OS/400 PASE C 코드

다음의 코드 예에는 코드를 설명하는 주석이 여러 곳에 나와 있습니다. 예를 입력하거나 검토할 때 이 주석을 반드시 읽어주십시오.

```
/* 이름: PASEtoILE.c
 *
 * 컴파일러 옵션 -qalign=natural 및 -qldbl128을 사용하여
 * 상대 16바이트 정렬 유형을 long double(유형 ILEpointer
 * 내부에서 사용됨)로 만들어야 합니다.
 */

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

/*
 * init_pid는 ILEtarget에서 주소 지정한 ILEpointer를
 * 추출한 프로세스의 프로세스 ID(PID)를 저장합니다.
 * init_pid는 이 프로그램의 exec() 이후에 나오는
 * 첫 번째 참조에서 초기화를 강제 수행하기 위한
 * 유효한 PID가 아닌 값으로 초기화됩니다.
 *
 * 사용자 코드가 pthread 인터페이스를 사용하는 경우
 * pthread_atfork()를 사용하여 등록된 핸들러를 제공하여
 * 하위 프로세스에서 ILE 프로시듀어 포인터를 다시
 * 초기화하고 정적 기억장치에서 포인터나 플래그를
 * 사용하여 exec() 이후 재초기화를 강제 수행할 수
 * 있습니다.
```

```
 */

pid_t  init_pid = -1;
ILEpointer*ILEtarget;  /* ILE 프로시듀어에 대한 포인터 */

/*
 * ROUND_QUAD는 지정된 주소나 그 주위에서 16바이트
 * 정렬 메모리 위치를 찾습니다.
 */

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

/*
 * do_init는 ILE 서비스 프로그램을 로드하고 해당
 * 서비스 프로그램에서 내보낸 프로시듀어로
 * ILE 포인터를 추출합니다.
 */

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

   /* _ILELOAD()는 서비스 프로그램을 로드합니다. */
   actmark = _ILELOAD("SHUPE/ILEPASE", ILELOAD_LIBOBJ);
   if (actmark == -1)
   abort();

   /*
     * xlc는 모든 유형의 자동(스택) 변수에 대해 16바이트
    * 정렬을 보장하지 않으므로 크기가 초과된
    * 버퍼에서 정렬된 영역을 찾습니다. _ILESYM()은
    * 서비스 프로그램 활성화에서 ILE 프로시듀어
    * 포인터를 추출합니다.
    */

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

   /*
    * 현재의 PID를 정적 기억장치에 저장하므로 (포크(fork))
    * 이후의 재초기화 시기를 판별할 수 있습니다.
    */
   init_pid = getpid();
}

/*
 * "aggregate"는 by-value 인수로 전달되는 구조나
 * 결합 자료 유형의 예입니다.
 */
typedef struct {
    char        filler[5];
} aggregate;

/*
```

```
 * "result_type" 및 "signature"는 ILEtarget에서
 * 식별하는 ILE 프로시듀어에 필요한 모든 인수의
 * 순서와 유형 및 함수 결과 유형을
 * 정의합니다.
 *
 * 주: 이 인수 리스트에 4바이트 미만의 고정 소수점
 * 인수 또는 8바이트 미만의 부동 소수점 인수가
 * 포함된다는 사실은 목표 ILE C 프로시듀어가
 * #pragma 인수(ileProcedureName, nowiden)를
 * 사용하여 컴파일됨을 의미합니다.
 *
 * 이 pragma가 없으면, ILE의 표준 C 연계에서
 * 1바이트 및 2바이트 정수 인수를 4바이트로
 * 확장해야 하고 4바이트 부동 소수점 인수를
 * 8바이트로 확장해야 합니다.
 */
static result_type_tresult_type = RESULT_INT32;
static arg_type_tsignature[] =
{
    ARG_INT32,
    ARG_MEMPTR,
    ARG_FLOAT64,
    ARG_UINT8,      /* ILE 코드에 #pragma nowiden이 있어야 합니다. */
    sizeof(aggregate),
    ARG_INT16,
    ARG_END
};

/*
 * wrapper_1은 자신이 호출하는 ILE 프로시듀어와 같은
 * 인수를 채택하여 같은 결과를 리턴합니다.
 * ILE 인수 리스트에 대해 사용자 정의되거나 선언된
 * 구조를 필요로 하지 않습니다. 이 랩퍼는 malloc을
 * 사용하여 기억장치를 확보합니다. 예외나 신호가
 * 발생하는 경우 기억장치를 해제하지 못할 수 있습니다.
 * 이러한 기억장치 누출 방지가 프로그램에 필요한 경우
 * 이를 처리할 신호 핸들러를 빌드하거나 wrapper_2에서
 * 메소드를 사용할 수 있습니다.
 */
int wrapper_1(int arg1, void *arg2, double arg3,
                        char arg4, aggregate arg5, short arg6)
{
  int result;
    /*
     * xlc는 모든 유형의 자동(스택) 변수에 대해 16바이트
     * 정렬을 보장하지 않지만 PASE malloc()은 반드시
     * 16바이트로 정렬된 기억장치를 리턴합니다.
     * size_ILEarglist()는 서명 배열의 항목에 기초하여
     * 필요한 기억장치의 용량을 판별합니다.
     */
    ILEarglist_base *ILEarglist;
    ILEarglist = (ILEarglist_base*)malloc( size_ILEarglist(signature) );

    /*
     * build_ILEarglist()는 신호 배열의 항목에
     * 기초하여 인수 값을 ILE 인수 리스트 버퍼에
     * 복사합니다.
     */
```

```
    build_ILEarglist(ILEarglist,
             &arg1,
             signature);

    /*
     * 저장된 PID 값을 사용하여 ILE 포인터가 설정되었는지
     * 여부를 검사할 수 있습니다. fork()의 하위 프로세서에서
     * 계승한 ILE 프로시듀어 포인터는 사용할 수 없는데,
     * 이들은 상위 프로세스에서 ILE 활성 그룹을 가리키기
     * 때문입니다.
     */
    if (getpid() != init_pid)
    do_init();

    /*
     * _ILECALL은 ILE 프로시듀어를 호출합니다. 예외나 신호가
     * 발생하는 경우 힙(heap) 할당이 분리됩니다(기억장치 누출).
     */
    _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;
}

/*
 * ILEarglistSt는 ILE 인수 리스트의 구조를 정의합니다.
 * xlc는 ILEpointer에 128비트 long double 멤버가 들어
 * 있기 때문에 ILEpointer 멤버 필드의 16바이트(상대)
 * 정렬을 제공합니다. 명시적 채움 필드는 자연적으로
 * ILE 관리 경계 내에 들지 않는 구조와 결합 유형의
 * 앞에서만 필요합니다.
 */
typedef struct {
    ILEarglist_base base;
    int32 arg1;
    /* 컴파일러가 제공하는 내재적 12바이트 채움 */
    ILEpointer arg2;
    float64 arg3;
    uint8 arg4;
    char filler[7]; /* 8바이트 정렬에 맞춰 채움 */
    aggregate arg5; /* 5바이트 집합(8바이트 정렬) */
    /* 컴파일러가 제공하는 내재적 1바이트 채움 */
    int16 arg6;
} ILEarglistSt;

/*
 * wrapper_2는 자신이 호출하는 ILE 프로시듀어와 같은
 * 인수를 채택하여 같은 결과를 리턴합니다.
 * 이 방법에서는 ILE 인수 리스트에 대해 사용자 정의된
 * 또는 선언된 구조를 사용하여 실행 효율을 높이고
```

```
 * 예외나 신호 발생 시 힙(heap) 기억장치 누출을 방지합니다.
 */
int wrapper_2(int arg1, void *arg2, double arg3,
                        char arg4, aggregate arg5, short arg6)
{
    /*
     * xlc는 모든 유형의 자동(스택) 변수에 대해 16바이트
     * 정렬을 보장하지 않으므로 크기가 초과된 버퍼에서
     * 정렬된 영역을 찾습니다.
     */
    char ILEarglist_buf[sizeof(ILEarglistSt) + 15];
    ILEarglistSt *ILEarglist = (ILEarglistSt*)ROUND_QUAD(ILEarglist_buf);
    /*
     * 지정문이 build_ILEarglist()를 호출하는 것보다
     * 빠릅니다.
     */
    ILEarglist->arg1 = arg1;
    ILEarglist->arg2.s.addr = (address64_t)arg2;
    ILEarglist->arg3 = arg3;
    ILEarglist->arg4 = arg4;
    ILEarglist->arg5 = arg5;
    ILEarglist->arg6 = arg6;
    /*
     * 저장된 PID 값을 사용하여 ILE 포인터가 설정되었는지
     * 여부를 검사할 수 있습니다. fork()의 하위 프로세서에서
     * 계승한 ILE 프로시듀어 포인터는 사용할 수 없는데,
     * 이들은 상위 프로세스에서 ILE 활성 그룹을 가리키기
     * 때문입니다.
     */
    if (getpid() != init_pid)
    do_init();
    /*
     * _ILECALL은 ILE 프로시듀어를 호출합니다. 스택은
     * 영향을 받지 않지만 예외나 신호 발생 시 힙(heap)
     * 기억장치가 분리되지 않습니다.
     */
    _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 코드

이제 OS/400 시스템에서 이 예에 대한 ILE C 코드를 작성하십시오. 코드를 작성할 라이브러리에서 소스 실제 파일이 필요합니다. ILE 예에는 주석이 곳곳에 분포합니다. 이들 주석은 코드를 이해하는 데 있어 아주 중요합니다. 소스를 입력하거나 검토할 때 이 주석을 반드시 검토하십시오.

```
#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) /* 불필요 */

/*
 * 이 ILE 프로시듀어에 대한 인수와 함수 결과는 OS/400 PASE
 * 프로그램에서 _ILECALL 함수에 제공되는 값과 같아야
 * 합니다.
 */
int ileProcedure(int         arg1,
                 char        *arg2,
                 double      arg3,
                 char        arg4[2],
                 aggregate   arg5,
                 short       arg6)
{
    char         fromcode[33];
    char         tocode[33];
    iconv_t      cd;     /* 변환 설명자 */
    char         *src;
    char         *tgt;
    size_t       srcLen;
    size_t       tgtLen;
  int result;

    /*
     * 변환 설명자를 열어 CCSID 37(EBCDIC)을, 호출자에
     * 리턴되는 문자 자료에 사용되는 CCSID 819(ASCII)로
     * 변환하십시오.
     */
    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;
 }
  /*
   * arg1이 1이면 (ASCII0로 변환된) 상수 텍스트를
   * arg2가 주소 지정하는 버퍼에 리턴하십시오. 다른
   * 모든 arg1 값에 대해서는 파일을 열고 텍스트를
   * 읽은 다음 (ASCII로 변환된) 해당 텍스트를 arg2가
   * 주소 지정하는 버퍼에 리턴하십시오.
   */
 if (arg1 == 1)
 {
     src = "Sample 1 output text";
     srcLen = strlen(src) + 1;
          tgt = arg2; /* arg2 버퍼로 iconv 출력 */
          tgtLen = srcLen;
          iconv(cd, &src, &srcLen, &tgt, &tgtLen);

     result = 1;
 }
else
 {
     FILE *fp;
     fp = fopen("SHUPE/PASEDATA", "r");
     if (!fp) /* 파일 열기 오류 시 */
     {
          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; /* 널(null) 종료된 빈 버퍼 */
          }
          src = buf;
          srcLen = strlen(buf) + 1;
          tgt = arg2; /* arg2 버퍼로 iconv 출력 */
          tgtLen = srcLen;
          iconv(cd, &src, &srcLen, &tgt, &tgtLen);

          fclose(fp);
     }
     result = 1;
 }
   /*
```

```
       * 변환 설명자를 닫고 위에서 판별된
       * 결과 값을 리턴하십시오.
       */
      iconv_close(cd);
      return result;
}
```

프로그램을 작성하기 위한 컴파일러 명령

OS/400 PASE 프로그램을 컴파일하는 경우 컴파일러 옵션 -qalign=natural과 -qldbl128을 사용하여 상대 16바이트 정렬 유형을 유형 ILEpointer 내부에서 사용되는 long double로 만들어야 합니다. 이 정렬은 OS/400의 ILE에 필요합니다. 옵션 -bI:에 대해서는 as400_libc.exp를 저장한 경로명을 입력해야 합니다.

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

ILE C 모듈과 서비스 프로그램을 컴파일하는 경우 teraspace 옵션을 사용하여 이들을 컴파일하십시오. 그렇지 않으면 OS/400 PASE가 이들과 대화할 수 없습니다.

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

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

마지막으로 DDS를 컴파일하고 적어도 하나의 자료 레코드를 전파해야 합니다.

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

OS/400 PASE에서 OS/400 프로그램 호출

OS/400 PASE 어플리케이션 작성 시 기존 OS/400 프로그램(*PGM 오브젝트)을 활용할 수 있습니다.

- systemCL 함수를 사용하여 CL CALL 명령을 실행하십시오. 이에 대한 정보와 예에 대해서는 OS/400 PASE에서 OS/400 명령 실행을 참조하십시오.
- _PGMCALL 런타임 함수를 사용하여 OS/400 PASE 프로그램 내에서 OS/400 프로그램을 호출할 수 있습니다. 이 방법은 systemCL 런타임 함수보다 실행 속도는 더 빠르지만 문자 스트링 인수의 자동 변환을 수행하지 않으며 다른 작업에서 프로그램을 호출할 수 있는 기능은 제공하지 않습니다.

_PGMCALL 런타임 함수를 사용하여 OS/400 PASE 프로그램에서 명령을 호출하는 방법에 대해서는 예: OS/400 PASE에서 OS/400 프로그램 호출을 참조하십시오.

예: OS/400 PASE에서 OS/400 프로그램 호출: 다음 예(면책사항 참조)는 _PGMCALL 런타임 함수를 사용하여 OS/400 PASE 프로그램에서 프로그램을 호출하는 방법을 보여줍니다.

```
/* 이 예에서는 OS/400 PASE _PGMCALL 함수를 사용하여
   OS/400 API QSZRTVPR을 호출합니다.

   QSZRTVPR API는 OS/400 소프트웨어 제품 로드에 대한 정보를
   검색하는 데 사용됩니다. API를 호출하는 데 필요한 입력 및
   출력 매개변수에 관한 특정 정보는 QSZRTVPR API 문서를
   참조하십시오. */

#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(QSZRTVPR 등)는 보통 EBCDIC인 문자 매개변수를
      기대합니다. 그러나 OS/400 PASE 프로그램의 문자 상수는
      보통 ASCII입니다. 그러므로 QSZRTVPR을 호출하는 데
      필요한 일부 CCSID 37(EBCDIC) 문자 매개변수 상수를
      선언하십시오. */

   /* format[]은 QSZRTVPR에 대한 입력 매개변수 3이고
      EBCDIC인 텍스트 'PRDR0100'으로 초기화됩니다. */
   const char format[] =
      {0xd7, 0xd9, 0xc4, 0xd9, 0xf0, 0xf1, 0xf0, 0xf0};

   /* prodinfo[]는 QSZRTVPR에 대한 입력 매개변수 4이고
      EBCDIC인 텍스트 '*OPSYS *CUR  0033*CODE'로 초기화됩니다.

      이 값은 현재 설치된 OS/400 릴리스의 옵션 33에 대한
      코드 로드를 검사하고자 함을 나타냅니다. */
   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};

   /* installed는 QSZRTVPR에서 리턴한 정보의 "로드 상태"
      필드와 비교되고 EBCDIC인 텍스트 '90'으로
      초기화됩니다. */
   const char installed[] = {0xf9, 0xf0};

   /* rcvr은 QSZRTVPR의 출력 매개변수 1입니다. */
   char rcvr[108];

   /* rcvrlen은 QSZRTVPR에 대한 입력 매개변수 2입니다. */
   int rcvrlen = sizeof(rcvr);

   /* errcode는 QSZRTVPR에 대한 입력 매개변수 5입니다. */
   struct {
      int bytes_provided;
      int bytes_available;
      char msgid[7];
   } errcode;


   /* qszrtvpr_pointer에는 QSZRTVPR에 대한 OS/400 16바이트 태그
```

```
   시스템 포인터가 들어 갑니다. */
  ILEpointer qszrtvpr_pointer;

  /* qszrtvpr_argv6은 QSZRTVPR에 대한 인수 포인터의 배열입니다. */
  void *qszrtvpr_argv[6];

  /* _RSLOBJ2 및 _PGMCALL 함수의 리턴 코드 */
 int rc;

  /* OS/400 포인터를 QSYS/QSZRTVPR *PGM 오브젝트로 설정하십시오. */
  rc = _RSLOBJ2(&qszrtvpr_pointer,
                RSLOBJ_TS_PGM,
                "QSZRTVPR",
                "QSYS");

  /* QSZRTVPR 리턴 정보 구조를 초기화하십시오. */
  memset(rcvr, 0, sizeof(rcvr));

  /* QSZRTVPR 오류 코드 구조를 초기화하십시오. */
  memset(&errcode, 0, sizeof(errcode));
  errcode.bytes_provided = sizeof(errcode);

  /* QSZRTVPR API에 대한 인수 포인터의 배열을 초기화하십시오. */
  qszrtvpr_argv[0] = &rcvr;
  qszrtvpr_argv[1] = &rcvrlen;
  qszrtvpr_argv[2] = &format;
  qszrtvpr_argv[3] = &prodinfo;
  qszrtvpr_argv[4] = &errcode;
  qszrtvpr_argv[5] = NULL;

  /* OS/400 PASE에서 OS/400 QSZRTVPR API를 호출하십시오. */
  rc = _PGMCALL(&qszrtvpr_pointer,
                (void*)&qszrtvpr_argv,
                0);

  /* 리턴된 정보의 63-64바이트에 대한 내용을 검사하십시오.
     내용이 '90'(EBCDIC)이 아닐 경우 코드 로드가 정확히
     설치되지 않은 것입니다. */
  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);
}
```

OS/400 PASE에서 OS/400 명령 실행

OS/400 기능을 사용하는 제어 언어(CL) 명령을 실행하여 OS/400 PASE 프로그램의 성능을 확장할 수 있습니다. OS/400 PASE 프로그램 내에서 OS/400 명령을 실행하려면 systemCL 런타임 함수를 사용하십시오.

OS/400 PASE에서 OS/400 명령을 실행하는 경우 systemCL 런타임 함수가 자동으로 문자 스트링 인수의 ASCII 대 EBCDIC 변환을 자동으로 처리하여 사용자가 다른 작업에서 프로그램을 호출할 수 있도록 해줍니다.

OS/400 PASE 프로그램에서 CL 명령을 실행하는 방법에 대한 예는 예: OS/400 PASE에서 OS/400 명령 실행을 참조하십시오.

예: OS/400 PASE에서 OS/400 명령 실행: 다음 예는(면책사항 참조) OS/400 PASE 프로그램에서 명령을 호출하는 방법을 보여줍니다.

```
/* sampleCL.c

   sampleCL을 사용한 CL 명령의 실행 예

   다음과 유사한 명령을 사용하여 컴파일하십시오.
   xlc -o sampleCL -I /whatever/pase -bI:/whatever/pase/as400_libc.exp sampleCL.c

   다음은 QP2SHELL()을 사용한 프로그램 예입니다.
   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]);

   /* CL 명령을 처리합니다 */
   rc = systemCL(argv[1], /* CL 명령의 첫 번째 매개변수를 사용합니다 */
                 SYSTEMCL_MSG_STDOUT
                 SYSTEMCL_MSG_STDERR ); /* 메세지를 수집합니다 */

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

OS/400 PASE 프로그램과 OS/400의 대화 방식

OS/400 PASE 프로그램이 OS/400 기능을 사용하도록 사용자 정의할 때 사용자 프로그램이 이들 기능과 대화하는 방식을 고려해야 합니다. 다음 주제에서는 기본 안내를 제공하고 iSeries Information Center의 자세한 OS/400 시스템 정보에 대한 링크를 제공합니다.

- 통신
- 데이터베이스
- 자료 코드화
- 파일 시스템
- 국제화
- 메세지 서비스
- 인쇄
- 유사 단말기(PTY)
- 보안
- 작업 관리

통신

OS/400 PASE는 소켓 통신에 대해 AIX와 동일한 구문을 지원합니다. 모든 세부사항에서 다른 UNIX(TM) 시스템과 일치하지 않을 수 있습니다.

OS/400 PASE 소켓 지원은 AIX의 소켓 구현과 유사하지만 OS/400 PASE는 OS/400의 소켓 구현을 사용하는데(AIX 커널의 소켓 구현이 아님) 이로 인해 AIX에서와 작동 방식에서 약간의 차이가 발생합니다.

OS/400의 소켓 구현은 UNIX 98과 BSD(Berkeley Software Distributions) 소켓을 모두 지원합니다. 대부분의 경우 OS/400 PASE는 AIX 구현의 작동을 일시적으로 허용하여 스타일에서의 차이를 해결합니다.

또한 실행 중인 어플리케이션에 대한 사용자 프로파일에는 소켓 API에서 level 매개변수를 IPPROTO_IP로 지정하고 option_value 매개변수를 IP_OPTIONS로 지정하기 위한 *IOSYSCFG 특수 권한이 있어야 합니다. OS/400에서의 소켓 사용에 대한 자세한 내용은 소켓 프로그래밍 주제를 참조하십시오. 특히 BSD(Berkeley Software Distributions) 호환성 및 UNIX 98 호환성 주제를 참조하십시오.

데이터베이스

OS/400 PASE는 iSeries용 DB2 UDB 호출 레벨 인터페이스(CLI)를 지원합니다. AIX와 OS/400의 DB2 CLI는 서로에 대해 적합한 서브세트가 아니기 때문에 다수의 인터페이스에서 약간의 차이가 있고 구현된 일부 API가 다른 시스템에는 없을 수 있습니다. 이 때문에 다음 사항을 고려해야 합니다.

- 코드를 생성할 수는 있지만 AIX 자체에서는 테스트할 수 없습니다. 대신 OS/400 PASE 내의 여러 플랫폼에서 코드를 테스트해야 합니다.

- »

  헤더 파일의 OS/400 버전 sqlcli.h를 사용하여 컴파일해야 합니다. 이 헤더 파일의 AIX 버전을 사용하여 컴파일된 프로그램은 OS/400 PASE에서 실행되지 않습니다.

  «

OS/400은 기본적으로 EBCDIC으로 코드화된 시스템인 반면 AIX는 ASCII에 기반합니다. 이러한 차이 때문에 OS/400 데이터베이스(iSeries용 DB2 UDB)와 OS/400 PASE 어플리케이션 사이에서 자료 변환이 필요한 경우가 종종 있습니다.

DB2 CLI를 OS/400 PASE에서 구현하는 경우 OS/400 PASE 제공 라이브러리 루틴은 자동으로 문자 자료에 대해 ASCII에서 EBCDIC으로 또는 그 반대로 자료 변환을 수행합니다. 변환은 액세스 중인 자료의 태그 CCSID(코드화 문자 세트 ID)와 OS/400 PASE 프로그램을 실행하는 ASCII CCSID에 기초하여 이루어집니다. 데이터베이스가 CCSID 65535를 사용하여 태그되는 경우 자동 변환은 수행되지 않습니다. 자료의 코드화 형식을 이해하고 필요한 변환을 수행하는 것은 어플리케이션의 몫입니다.

CCSID에 대한 작업

Qp2RunPase() API를 사용하는 경우 OS/400 PASE CCSID를 명시적으로 지정해야 합니다.

»

API 프로그램 QP2TERM, QP2SHELL 또는 QP2SHELL2를 호출하기 전에 ILE 환경에서 다음 변수를 모두 설정하여 OS/400 PASE CCSID를 제어할 수 있습니다.

- PASE_LANG
- QIBM_PASE_CCSID

ILE 환경에 이들 변수가 하나 또는 모두 누락된 경우 QP2TERM, QP2SHELL 및 QP2SHELL2는 기본적으로 사용자 작업의 언어와 CCSID 속성에 가장 적합한 OS/400 PASE 값으로 OS/400 PASE CCSID 및 OS/400 PASE 환경 변수 LANG을 설정합니다.

«

자세한 정보는 QP2TERM() 및 QP2SHELL() 프로그램 설명을 참조하십시오.

»

libc.a의 확장으로 OS/400 PASE 어플리케이션은 _SETCCSID() 함수를 사용하여 어플리케이션의 실행 중인 CCSID를 변경할 수 있습니다.

또다른 확장으로 OS/400 PASE 어플리케이션은 어플리케이션의 CCSID를 변경하지 않고도 DB2 CLI 내부 변환을 대체할 수 있습니다. SQLOverrideCCSID400() 함수는 대체 CCSID의 정수를 단일 매개변수로 받아

들입니다.

주: CCSID 대체 함수 SQLOverrideCCSID400()가 효력을 가지려면 다른 SQLx() API 이전에 이를 호출해야 합니다. 그렇지 않으면 요청이 무시됩니다.

OS/400 PASE 프로그램에서 iSeries용 DB2 UDB CLI 사용

OS/400 PASE 프로그램에서 DB2 CLI를 사용하려면 소스를 컴파일하기 전에 sqlcli.h 헤더 파일 및 libdb400.exp 내보내기 파일을 사용자의 AIX 시스템으로 복사해야 합니다. DB2 CLI 라이브러리 루틴은 OS/400 PASE 환경의 경우 libdb400.a에 있으며 pthread 인터페이스를 사용하여 구현되어 스레드세이프를 제공합니다.

대다수의 OS/400 PASE CLI 함수는 원하는 연산을 수행하기 위해 상응하는 ILE CLI 함수를 호출합니다.

DB2 UDB 호출 레벨 인터페이스에 대한 자세한 정보는 iSeries용 DB2 UDB SQL 호출 레벨 인터페이스 (ODBC) 주제를 참조하십시오.

OS/400 PASE가 iSeries용 DB2 UDB SQL 호출 레벨 인터페이스를 사용하여 iSeries용 DB2 UDB에 액세스하는 방법의 예에 대해서는
예: OS/400 PASE 프로그램에서 iSeries용 DB2 UDB CLI 함수 호출을 참조하십시오.

예: OS/400 PASE 프로그램에서 iSeries용 DB2 UDB CLI 함수 호출: 다음 예(면책사항 참조)는 iSeries용 DB2 UDB SQL 호출 레벨 인터페이스를 사용하여 iSeries용 DB2 UDB에 액세스하는 OS/400 PASE 프로그램을 보여줍니다.

```
/* OS/400 PASE iSeries용 DB2 UDB 프로그램 예
 *
 * SQL CLI를 통해 OS/400 DB2 UDB에 액세스하는
 * OS/400 PASE 프로그램의 예를 보여줍니다.
 *
 * 프로그램은 모든 시스템에 존재해야 하는 iSeries Access 데이터베이스,
 * QIWS/QCUSTCDT에 액세스합니다.
 *
 * fun_Connect() 프로시듀어의 시스템명, 사용자 ID 및 암호를
 * 유효한 매개변수로 변경하십시오.
 *
 * 컴파일 호출:
 *
 * xlc -I./include -bI:./include/libdb400.exp -o paseclidb4 paseclidb4.c
 *
 * 2진으로 FTP 전송하고 QP2TERM() 단말기 쉘에서 실행하십시오.
 *
 * 출력에는 STATE 컬럼이 MN인 모든 행이 표시되어야 합니다.
 */
```

```
/* 변경 활동: */
/* 변경 활동 끝 */

#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 */
```

자료 코드화

대부분의 UNIX 시스템은 ASCII 문자 코드화를 사용합니다.

대부분의 OS/400 함수는 EBCDIC 문자 코드화를 사용합니다. 일부 OS/400 오브젝트 유형에 대해 CCSID(코드화 문자 세트 ID)를 지정하여 오브젝트의 문자 자료에 고유한 코드화를 식별할 수 있습니다.

OS/400 PASE 바이트 스트림 파일은 OS/400 PASE 외부의 대다수 시스템 인터페이스가 파일에 쓰거나 파일에서 읽어오는 텍스트 자료를 필요한 대로 변환할 때 사용하는 CCSID 속성을 가지고 있습니다. OS/400 PASE는 스트림 파일에 쓰거나 이 파일에서 읽어온 자료에 대해 CCSID 변환을 수행하지 않지만(AIX와 일치함) 다른 시스템 함수가 파일의 ASCII 텍스트를 올바로 처리할 수 있도록, OS/400 PASE가 작성하는 바이트 스트림 파일의 CCSID 속성을 현재의 OS/400 PASE 값으로 설정합니다.

OS/400 PASE 공유 라이브러리에 들어 있는 AIX API를 사용하는 경우 OS/400 PASE가 대부분의 자료 변환을 처리합니다.

OS/400 PASE 프로그램은 OS/400 PASE 런타임에서 자동으로 처리하지 않는 문자 자료 변환에 대해서는 공유 라이브러리 `libiconv.a`에 제공되는 `iconv` 함수를 사용할 수 있습니다. 예를 들어 OS/400 PASE 어플리케이션은 보통(_ILECALLX 또는 _PGMCALL를 사용하여) OS/400 API 함수를 호출하기 전에 문자 스트링을 EBCDIC으로 변환해야 합니다.

파일 시스템

OS/400 PASE 프로그램은 QSYS.LIB 및 QOPT 파일 시스템에 있는 오브젝트를 포함하여 통합 파일 시스템을 통해 액세스할 수 있는 모든 파일이나 자원에 액세스할 수 있습니다.

버퍼링되는 입력 및 출력

외부 장치와의 사이에서 발생하는 입력 및 출력은 OS/400에 버퍼링됩니다. 자료 블록을 처리하는 입력 및 출력 프로세서가 이를 처리합니다. 이와 반대로 UNIX 시스템은 보통 문자 단위(버퍼링되지 않은) 입출력 방식으로 작동합니다. OS/400에서는 특정한 입력 및 출력 신호(예: Enter 키, 기능 키 및 시스템 요청)만이 시스템에 인터럽트를 전송할 수 있습니다.

자료 변환 지원

OS/400 PASE 프로그램은 ASCII(또는 UTF-8) 경로명을 `open` 함수에 전달하여 바이트 스트림 파일을 열고, 여기서 이름이 자동으로 OS/400 시스템이 사용하는 코드화 체계로 변환됩니다. 그러나 열린 파일에서 읽어 오거나 열린 파일에 쓰여진 자료는 변환되지 않습니다.

자료 변환에 대한 자세한 정보는 자료 코드화를 참조하십시오.

파일 설명자 사용

OS/400 PASE 런타임은 보통 `stdin`, `stdout` 및 `stderr` 파일에 대해 ILE C 런타임 지원을 사용하여 OS/400 PASE 및 ILE 프로그램에 일관된 작동을 제공합니다.

OS/400 PASE 및 ILE C는 표준 입력 및 출력(stdin, stdout 및 stderr)에 동일한 스트림을 사용합니다. OS/400 PASE 프로그램은 항상 파일 설명자 0, 1 및 2를 사용하여 표준 입력 및 출력에 액세스합니다. 그러나 ILE C는 stdin, stdout 및 stderr에 대해 항상 통합 파일 설명자를 사용하는 것이 아니므로 OS/400 PASE는 OS/400 PASE 파일 설명자와 통합 파일 시스템의 설명자 사이에서 맵핑을 제공합니다. 이러한 맵핑으로 인해 OS/400 PASE 프로그램과 ILE C 프로그램은 동일한 열린 파일에 액세스할 때 서로 다른 설명자 번호를 사용합니다.

`fcntl` 함수 F_MAP_XPFFD에서 OS/400 PASE 확장자를 사용하여 OS/400 PASE 설명자를 ILE 숫자에 할당할 수 있습니다.

»

이는 OS/400 PASE 어플리케이션이 OS/400 PASE에서 작성하지 않은 ILE 설명자에 대해 파일 조작을 수행해야 하는 경우 유용합니다.

«

`fstatx` 함수 STX_XPFFD_PASE에 대한 OS/400 고유 확장자를 이용하면 OS/400 PASE 프로그램은 OS/400 PASE 파일 설명자에 대한 통합 파일 시스템 설명자 번호를 판별할 수 있습니다. 파일 `stdin`, `stdout` 및 `stderr`에 대한 ILE C 런타임 지원에 접속된 OS/400 PASE 설명자에 대해 특수 값(음수)이 리턴됩니다.

Qp2RunPase() API가 호출될 때 ILE 환경 변수 QIBM_USE_DESCRIPTOR_STDIO가 Y 또는 I로 설정된 경우 OS/400 PASE는 파일 설명자 0, 1 및 2를 통합 파일 시스템과 동기화하여 OS/400 PASE와 ILE C 프로그램이 파일 `stdin`, `stdout` 및 `stderr`에 대해 동일한 설명자 번호를 사용하도록 합니다. 이 모드에서 작동 중일 때 OS/400 PASE 코드 또는 ILE C 코드가 파일 설명자 0, 1 또는 2를 닫거나 새로 열 경우 이러한 변경은 두 환경 모두에서 `stdin`, `stdout` 및 `stderr` 처리에 영향을 미칩니다.

OS/400 PASE 런타임은 보통 OS/400 PASE 파일 설명자(소켓 포함)를 통해 읽거나 쓴 자료에 대해서는 문자 코드화 변환을 수행하지 않습니다. 다만 ILE C stdin에서 읽어 온 자료나 ILE C stdout 및 stderr에 쓴 자료에 대해서는 (OS/400 PASE CCSID 및 작업 디폴트 CCSID 사이에서) ASCII 대 EBCDIC 변환이 수행됩니다.

두 개의 환경 변수가 stdin, stdout 및 stderr의 자동 변환을 제어합니다.

- 일반적으로 적용되는 변수는 QIBM_USE_DESCRIPTOR_STDIO입니다. Y로 설정되면 ILE 런타임은 이들 파일에 대해 파일 설명자 0, 1 또는 2를 사용합니다.
- PASE 고유 환경 변수는 QIBM_PASE_DESCRIPTOR_STDIO입니다. 이 변수는 2진에 대해서는 B의 값을 가지고 텍스트에 대해서는 T의 값을 갖습니다.

ILE 환경 변수 QIBM_USE_DESCRIPTOR_STDIO가 Y로 설정되고 QIBM_PASE_DESCRIPTOR_STDIO가 B로 설정된 경우(stdin에서 2진 자료를 읽어오고 stdout 또는 stderr에 2진 자료를 쓸 수 있음) OS/400 PASE stdin, stdout 및 stderr에 대한 ASCII 대 EBCDIC 변환은 작동 불가능합니다.
QIBM_PASE_DESCRIPTOR_STDIO에 대한 디폴트는 텍스트에 대한 T입니다. 이 값은 EBCDIC에서 ASCII로 변환이 이루어지게 합니다.

파일 시스템에 대한 자세한 정보는 통합 파일 시스템 주제를 참조하십시오.

국제화

OS/400 PASE 런타임은 AIX 런타임에 기반하고 있으므로, OS/400 PASE 프로그램은 AIX에서 지원되는 다수의 로케일, 문자 스트링 조작, 날짜 및 시간 서비스, 메세지 카탈로그 및 문자 코드화 변환에 대해 AIX에서와 같은 풍부한 프로그래밍 인터페이스를 사용할 수 있습니다.

OS/400 PASE는 단일 바이트와 다중 바이트 문자 코드화에 대한 지원 등 어플리케이션에서 사용하는 로케일을 관리하고 로케일 구분 함수(예: ctype 및 strcoll)를 수행하는 데 있어서 AIX 런타임의 인터페이스를 지원합니다.

OS/400 PASE에는 산업 표준 코드화(코드 세트 ISO8859-x), 코드 세트 IBM-1250 및 코드 세트 UTF-8을 사용하여 다수의 국가와 언어를 지원하는 AIX 로케일의 서브세트가 들어 있습니다.

OS/400 PASE는 IBM-1252 로케일 및 ISO 8859-15 로케일(모두 단일 바이트 코드화 사용), UTF-8 로케일의 세 가지 방식으로 유로를 지원합니다.

주: OS/400 PASE에 대한 로케일 지원은 ILE C 프로그램 (오브젝트 유형 *CLD 및 *LOCALE)에서 사용하는 로케일 지원 양식과는 무관합니다. 이러한 내부 구조의 차이 외에, ILE C 프로그램에 대해 기존에 제공된 로케일은 ASCII를 지원하지 않습니다.

새 로케일 작성

OS/400 PASE는 새 로케일 작성을 위한 유틸리티를 제공하지 않습니다. 그러나 `localedef` 유틸리티를 사용하면 AIX 시스템상의 OS/400 PASE에 사용할 로케일을 작성할 수 있습니다.

로케일 변경

OS/400 PASE 어플리케이션이 로케일을 변경하는 경우에는 새 로케일의 코드화와 일치시키기 위해 (_SETCCSID 런타임 함수를 사용하여) OS/400 PASE CCSID도 변경해야 합니다. 이렇게 하면 OS/400 PASE 런타임이 문자 자료 인터페이스 인수를 정확하게 해석할 수 있습니다(또한 EBCDIC 시스템 서비스 호출 시에도 적절히 변환할 수 있습니다). `cstoccsid` 런타임 함수를 사용하여 코드 세트 이름에 해당하는 CCSID를 판별할 수 있습니다.

OS/400 PASE 런타임은 OS/400 PASE 프로그램이 작성한 파일의 CCSID 태그를 현재의 OS/400 PASE CCSID 값(프로그램 시작 시 또는 가장 최근의 _SETCCSID 값을 사용하여 제공됨)으로 설정합니다.

일본어, 한국어, 정체 한자 및 간체 한자를 지원하는 OS/400 PASE 어플리케이션에는 UTF-8 로케일을 사용해야 합니다. OS/400에는 이들 언어를 위한 다른 로케일도 들어 있지만 시스템이 IBM-euc*XX* 코드 세트에 대한 코드화와 일치하도록 OS/400 PASE CCSID를 설정하는 것을 지원하지 않습니다. UTF-8 지원을 사용하면 다른 플랫폼에서 어플리케이션이 수행될 경우 다른 코드화(예: Shift-JIS)로 저장된 파일 자료를 변환해야 할 수 있습니다.

OS/400 PASE 변환 오브젝트와 로케일의 저장 위치

OS/400 PASE에 대한 변환 오브젝트와 로케일은 OS/400 언어 피처 코드와 함께 들어 있습니다. OS/400 PASE 설치 시 설치된 OS/400 언어 피처와 연관된 로케일만이 작성됩니다.

모든 OS/400 PASE 로케일이 ASCII 또는 UTF-8 문자 코드화를 사용하므로 모든 OS/400 PASE 런타임은 ASCII(또는 UTF-8)로 작동합니다.

OS/400에서의 국제화에 대한 자세한 정보는 국제화 주제를 참조하십시오.

메세지 서비스

OS/400 PASE 신호와 ILE 신호는 서로 독립적이므로 다른 유형의 신호를 발생시켜 한 신호 유형에 대한 핸들러를 직접 호출하는 것은 불가능합니다. OS/400 PASE Qp2SignalPase() API를 사용하면 수신된 임의의

ILE 신호에 해당하는 OS/400 PASE 신호를 공개할 수 있습니다. QP2SHELL() 프로그램과 OS/400 PASE `fork()` 함수는 모든 ILE 신호를 해당하는 OS/400 PASE 신호에 맵핑하기 위한 핸들러를 반드시 설정합니다.

시스템은 Qp2RunPase, Qp2CallPase 또는 Qp2CallPase2 API를 실행 중인 호출의 프로그램 메세지 대기행렬에 송신된 OS/400 예외 메시지를 해당하는 OS/400 PASE 신호로 변환합니다. 따라서 OS/400 PASE 어플리케이션은 시스템이 변환한 OS/400 PASE 신호를 처리함으로써 모든 OS/400 예외를 처리할 수 있습니다.

OS/400 PASE는 OS/400 메세지 처리에 대한 직접 제어를 부여하는 다음의 런타임 함수를 제공합니다.

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

이들 함수에 대한 자세한 내용은 런타임 함수를 참조하십시오.

OS/400 메세지 지원

OS/400은 다양한 문맥으로 메시지 지원을 제공합니다.

작업 기록부
작업 기록부에는 실행 중이나 컴파일 중 OS/400이나 어플리케이션이 발행한 모든 메세지가 들어 있습니다. 작업 기록부를 살펴보려면 명령행에 DSPJOBLOG를 입력하십시오. 작업 기록부 표시 화면이 나타나면 F10 키와 Shift + F6을 누르십시오. 이들 키 조합을 누르면 모든 메세지 표시 화면이 표시되고 가장 최신 메세지로 설정됩니다. 특정 메세지의 세부사항을 보려면 커서를 해당 메세지로 이동하여 F1 키를 누르십시오.

사용 중인 작업에 대한 작업
사용 중인 작업에 대한 작업(WRKACTJOB) 명령은 OS/400의 작업과 작업 스택을 검토하는 데 유용합니다.

OS/400에서의 메세지 지원에 대한 자세한 정보는 작업 관리 주제를 참조하십시오.

OS/400 PASE 및 OS/400 메세지에 대한 자세한 정보는 OS/400 PASE 신호 처리를 참조하십시오.

OS/400 PASE 어플리케이션에서 출력 인쇄

QShell Rfile 유틸리티를 사용하여 OS/400 PASE 쉘에서 출력을 읽고 쓸 수 있습니다.

다음 예는 스트림 파일 mydoc.ps의 내용을 변환되지 않은 ASCII 자료로 스풀 프린터 장치 파일 QPRINT에 기록한 다음 CL LPR 명령을 사용하여 스풀 파일을 다른 시스템에 송신합니다.

```
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
```

유사 단말기(PTY)

OS/400 PASE는 AT&T 및 BSD(Berkeley Software Distributions) 스타일 장치를 지원합니다. 프로그래밍 관점에서 보면 이들 장치는 AIX에서와 같은 방식으로 OS/400 PASE에서 작동합니다.

OS/400 PASE는 AT&T 스타일 장치에 대해 최대 1024개의 인스턴스를 지원하고 최대 592개의 BSD 스타일 장치를 지원합니다. 시스템이 시작되면 각 장치 유형의 처음 32개 인스턴스가 자동으로 작성됩니다.

OS/400 PASE에서 PTY 장치 구성

AIX에서 관리자는 smit를 사용하여 각 유형의 사용 가능한 장치 수를 구성합니다. OS/400 PASE에서는 다음과 같은 방법으로 이들 장치를 구성합니다.

- AT&T 스타일 장치의 경우 OS/400 PASE는 자동 구성을 지원합니다. 처음 32개 인스턴스가 사용 중이고 어플리케이션이 다른 인스턴스를 열려고 하면 최대 1024개의 한도까지 통합 파일 시스템에 CHRSF 장치가 자동으로 작성됩니다.
- BSD 스타일 장치의 경우 OS/400 PASE mknod 유틸리티를 사용하여 CHRSF 장치를 수동으로 작성해야 합니다. 이를 수행하려면 명명 규칙뿐 아니라 BSD 슬레이브 및 BSD 마스터 장치의 메이저 번호를 알아야 합니다. 다음 예의 쉘 스크립트는 추가의 BSD PTY 장치를 작성하는 방법을 보여줍니다. 16개 그룹으로 이 장치를 작성합니다.

```
#!/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
```

유사 단말기 장치에 대한 자세한 내용은 AIX문서 웹 사이트를 참조하십시오.

«

보안

보안상의 관점에서 OS/400 PASE 프로그램은 OS/400의 다른 프로그램과 동일한 보안 제한사항을 따르도록 되어 있습니다. OS/400에서 OS/400 PASE 프로그램을 실행하려면 통합 파일 시스템에서 AIX 2진(binary)에

대한 권한이 있어야 합니다. 또한 사용자 프로그램이 액세스하는 각 자원에 대해 적절한 레벨의 권한이 있어야 하고 그렇지 않으면 이 자원에 액세스하려 할 때 오류가 수신됩니다.

다음 정보는 OS/400 PASE 프로그램을 실행할 때 특히 중요합니다.

사용자 프로파일 및 권한 관리

시스템 권한 관리는 오브젝트이기도 한 사용자 프로파일에 기반합니다. 시스템에 작성되는 모든 오브젝트는 특정 사용자가 소유합니다. 오브젝트에 대한 각 조작이나 액세스는 사용자의 권한을 확인하기 위해 시스템에서 검증합니다. 소유자 또는 적절한 권한이 있는 사용자 프로파일은 오브젝트 조작을 위한 여러 가지 권한 유형을 다른 사용자 프로파일에 위임할 수 있습니다. 모든 유형의 오브젝트에 일정하게 권한 검사가 제공됩니다.

오브젝트 권한 메카니즘은 여러 가지 제어 레벨을 제공합니다. 사용자의 권한은 정확하게 필요한 것만으로 제한될 수 있습니다. QOpenSys 파일 시스템에 저장된 파일은 UNIX 파일과 같은 권한이 부여됩니다. 다음 표는 OS/400 데이터베이스 파일에 사용된 보안 값과 UNIX 권한 사이의 관계를 보여줍니다. OS/400에서 *OBJOPR은 *오브젝트 사용* 권한입니다. *EXCLUDE는 *권한 없음*입니다. *READ, *ADD, *UPD, *DLT 및 *EXECUTE는 자료 권한입니다. 파일을 OS/400 PASE 프로그램으로 실행하려면 파일에 *EXECUTE 권한(및 종종 *READ 권한)이 있어야 합니다.

UNIX 권한	*OBJOPR	*READ	*ADD	*UPD	*DLT	*EXECUTE
r(read)	X	X	-	-	-	-
w(write)	X	-	X	X	X	-
x(execute)	X	-	-	-	-	X
권한 없음	-	-	-	-	-	-

OS/400 PASE의 사용자 프로파일

OS/400에서 인증 정보는 /etc/passwd와 같은 파일이 아니라 개별 *프로파일*에 저장됩니다. 사용자와 그룹은 프로파일을 갖습니다. 이들 모든 프로파일이 이름공간을 공유하며 각 프로파일은 대소문자가 혼합되지 않은 고유한 이름을 가져야 합니다. 소문자 이름을 getpwnam() 또는 getgrnam() API에 전달하는 경우 시스템은 이름 스트링을 기대 값(소문자 또는 대문자)으로 변환합니다.

프로파일 이름을 리턴받기 위해 getpwuid() 또는 getgrgid()를 호출하는 경우 결과를 대문자로 리턴하는 OS/400 PASE 환경 변수 PASE_USRGRP_LOWERCASE=N를 설정하지 않았다면 프로파일 이름은 소문자입니다.

모든 사용자는 사용자 ID(UID)를 갖습니다. 모든 그룹은 그룹 ID (GID)를 갖습니다. 이들은 POSIX 1003.1 표준에 따라 정의됩니다. 두 개의 숫자 공간이 분리되므로 UID가 104인 사용자와 GID가 104인 그룹을 가질 수 있습니다.

OS/400은 0의 UID를 갖는 보안 담당자 QSECOFR에 대한 사용자 프로파일을 갖습니다. 다른 사용자 프로파일은 0의 UID를 가질 수 없습니다. QSECOFR는 시스템에서 가장 많은 특권을 가진 프로파일이며 그런 이유로 루트 사용자로 작용합니다. 그러나 OS/400은 시스템 관리자가 개별 사용자에게 할당할 수 있는 특정의 권한 세트도 제공합니다. 이러한 권한 중 하나인 *ALLOBJ는 파일 액세스에 대한 무조건적 액세스 제어(UNIX 시스템에서 루트 권한의 일반적 쓰임)를 대체합니다.

루트 액세스를 사용하는 이식된 어플리케이션에서는 *ALLOBJ 권한을 부여할 수 있는 *어플리케이션 사용자*에 대해 특정의 사용자 프로파일을 작성하여, 단일 어플리케이션이 필요로 하는 것보다 훨씬 많은 권한을 갖는 QSECOFR의 사용을 방지하는 것이 더 좋은 보안 습관입니다. UNIX 시스템과 달리 OS/400은 사용자에 대해 그룹 멤버쉽을 요구하지 않습니다. OS/400에서 사용자 프로파일에 대한 0의 GID는 더 많은 권한을 가진 그룹을 지칭하는 것이 아니라 *지정된 그룹 없음*을 의미합니다.

OS/400 보안은 시스템에 빌드된 통합 보안에 의존합니다. 오브젝트에 대한 모든 액세스는 보안 검사를 통과해야 합니다. 보안 검사는 액세스 시 프로세스가 실행 중이던 사용자 프로파일에 대해 수행됩니다.

OS/400 PASE는 무결성과 보안을 유지보수하기 위해 각 프로세스에 별도의 주소 공간을 부여하고 있습니다. 사용자의 OS/400 PASE 주소 공간에서 자원을 사용할 수 없는 경우, 이 자원에 액세스할 수 없습니다. 파일 시스템 보안은 누군가가 적절한 권한 없이 자신의 주소 공간으로 자원을 로드하는 것을 막아줍니다. 주소 공간에 있는 자원은 프로세스가 실행되는 ID에 관계 없이 프로세스에서 사용할 수 있습니다.

OS/400 PASE 프로그램은 시스템 호출을 사용하여 시스템 함수를 호출합니다. OS/400 PASE 프로그램에 대한 시스템 호출은 OS/400에서 처리합니다. 이 인터페이스는 시스템 내부에 대한 간접(및 보안) 액세스만을 OS/400 PASE 프로그램에 부여합니다.

iSeries 서버에서의 보안에 대해 자세히 알려면 보안 주제를 참조하십시오.

작업 관리

OS/400은 시스템의 다른 작업을 처리하는 것과 같은 방식으로 OS/400 PASE 프로그램을 처리합니다. OS/400의 작업 처리 방식에 대한 정보는 작업 관리 주제를 참조하십시오.

OS/400 PASE 문제 해결

이 정보는 OS/400 PASE 프로그램을 디버그하는 방법과 이 프로그램을 좀 더 효율적이고 효과적으로 실행하기 위한 안내를 제공합니다.

- OS/400 PASE 프로그램 디버그
- 성능 최적화

OS/400 PASE 프로그램 디버그

OS/400 PASE 런타임 환경은 `syslog()` 런타임 함수에 라이브러리 지원을 제공하고 (더 복잡한 메세지 라우팅을 위해) `syslogd` 2진(binary)을 제공합니다. 또한 OS/400의 기존 자원(진단 메세지 및 OS/400 시스템 오퍼레이터 메세지 대기행렬인 QSYSOPR에 심각한 메세지를 송신하기 위한 작업 기록부 등)을 사용할 수 있습니다.

어플리케이션에 따라 OS/400 PASE 어플리케이션을 디버깅하기 위한 전략은 경로가 달라질 수 있습니다.

- 어플리케이션이 OS/400 통합(예를 들어 iSeries용 DB2 UDB 또는 ILE 함수와의 통합)을 필요로 하지 않을 경우 먼저 AIX에서 어플리케이션을 디버그해야 합니다.
- 그런 후 OS/400 PASE dbx와 OS/400 디버그 기능의 조합(예: 작업 기록부)을 사용하여 OS/400에서 어플리케이션을 디버그하십시오.

데이터베이스 또는 ILE 함수를 사용하도록 코딩한 어플리케이션은 AIX에서 완전하게 테스트할 수 없지만 AIX에서 어플리케이션의 나머지 부분을 디버그하여 적절한 구조와 설계를 보장할 수 있습니다.

OS/400 PASE에서 dbx 사용

OS/400 PASE는 AIX dbx 디버거 유틸리티를 지원합니다. 이 유틸리티를 사용하면 적절히 컴파일되지 않은 경우 소스 코드 레벨에서 관련 프로세스(상위 및 하위)를 디버그할 수 있습니다. 네트워크 파일 시스템(NFS)을 사용하여 OS/400 PASE에서 실행되는 디버거에 AIX 소스가 보이게 만들 수 있습니다.

»

xterm 및 aixterm에 대한 OS/400 PASE 지원은 dbx를 사용하여 상위 및 하위 프로세스를 디버그할 수 있게 합니다. dbx는 dbx를 두 번째 프로세스에 접속한 상태로 다른 xterm 창을 시작합니다.

«

dbx에 대한 자세한 정보는 AIX 문서 웹 사이트를 참조하십시오. 또한 dbx 명령행에 help를 입력할 수도 있습니다.

OS/400 디버깅 툴 사용

ILE C 소스 디버거는 코드의 문제점을 판별하기 위한 효과적 툴입니다. 이 툴에 대해 배우려면 WebSphere(TM) Development Studio ILE C/C++ Programmer's Guide 를 참조하십시오.

»

OS/400 PASE 코드를 디버그하기 위해 iSeries 시스템 디버거를 사용할 수 있습니다.

«

성능 최적화

최상의 성능을 얻으려면 어플리케이션 2진(bianry)을 로컬 스트림 파일 시스템에 저장하도록 하십시오. 2진(기본 프로그램 및 라이브러리)이 로컬 스트림 파일 시스템 외부에 있을 경우 파일 맵핑을 수행할 수 없기 때문에 OS/400 PASE 프로그램을 시작하는 데 훨씬 오랜 시간이 걸립니다.

»

다수의 fork() 연산을 수행하는 어플리케이션을 OS/400 PASE에서 실행하는 경우 AIX에서처럼 빠르게 실행되지 않습니다. 각 OS/400 PASE fork() 연산이 새로운 OS/400 작업을 시작하기 때문인데 이는 성능에 중대한 영향을 미칠 수 있습니다.

성능 자료 수집 및 분석에 대한 정보는 시스템 관리 범주에서 성능 주제를 참조하십시오.

예

다음 예는 OS/400 PASE 정보로 제공되었습니다. 이 예를 사용하기 전에 아래의 "코드 예 면책사항"을 읽으십시오.

ILE 프로그램에서 OS/400 PASE 프로그램 및 프로시듀어 실행

- ILE 프로그램에서 OS/400 PASE 프로그램 실행
- ILE 프로그램에서 OS/400 PASE 프로시듀어 호출

OS/400 PASE 프로그램에서 OS/400 프로그램 호출

- OS/400 PASE 프로그램에서 ILE 프로시듀어 호출
- OS/400 PASE에서 OS/400 프로그램 호출
- OS/400 PASE에서 CL 명령 실행

OS/400 PASE 프로그램에서 iSeries용 DB2 UDB 함수 사용

- OS/400 PASE 프로그램에서 iSeries용 DB2 UDB 호출 레벨 인터페이스 호출

코드 예 면책사항

IBM은 귀하에게 유사한 기능을 귀하의 특정 요구에 맞게 조정하여 생성할 수 있도록 모든 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 사용권을 부여합니다.

모든 샘플 예제는 IBM에 의해 예시 목적으로만 제공됩니다. 이러한 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다. 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다.

여기에 포함된 모든 프로그램은 어떠한 종류의 보증없이 "현상태대로" 제공됩니다. 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증은 명백하게 제외됩니다.

OS/400 PASE 관련 정보

OS/400 PASE의 사용 방법에 대한 자세한 정보는 다음 자원을 참조하십시오.

기타 iSeries Information Center 주제

OS/400 PASE API

다음과 같은 OS/400 PASE API의 일반 범주에 대한 세부사항은 다음 주제를 참조하십시오.

- 호출 가능 프로그램 API
- ILE 프로시듀어 API
- OS/400 PASE 프로그램에서 사용하는 런타임 함수

OS/400 PASE 프로그램을 실행하려면 시스템 API를 호출해야 합니다. OS/400 PASE 프로그램을 실행하기 위한 호출 가능 프로그램 API와 ILE 프로시듀어 API를 모두 시스템에서 제공합니다. 호출 가능 프로그램 API가 사용하기는 더 쉽지만, ILE 프로시듀어 API에서 사용할 수 있는 모든 제어를 제공하지는 않습니다.

OS/400 PASE 쉘 및 유틸리티

OS/400 PASE에는 세 개의 쉘(Korn, Bourne 및 C 쉘)과 OS/400 PASE 프로그램으로 실행되는 약 200개의 유틸리티가 들어 있습니다. OS/400 PASE 쉘과 유틸리티는 다수의 산업 표준 및 사실상의 표준 명령을 포함하는 확장 가능한 스크립트 환경을 제공합니다.

OS/400 PASE 명령

이 주제에서 설명되는 대부분의 OS/400 PASE 명령은 AIX 명령과 동일한 옵션을 지원하며 동일한 작동을 제공합니다. OS/400 PASE 명령 외에 각 OS/400 PASE 쉘은 다수의 내장 명령(`cd`, `exec` 및 `if`)을 지원합니다.

OS/400 PASE 라이브러리

OS/400 PASE 런타임은 AIX 런타임이 제공하는 인터페이스의 대형 서브세트를 지원합니다. OS/400 PASE가 지원하는 대다수의 런타임 인터페이스는 AIX와 동일한 옵션과 작동을 제공합니다. OS/400 PASE 런타임 라이브러리는 `/usr/lib`에 기호 링크로 설치됩니다.

웹 사이트

OS/400 PASE가 지원하는 AIX 명령, API 및 유틸리티: API 분석 툴 은 사용자 어플리케이션이 사용하는 AIX 기능을 OS/400 PASE에서 지원하는 방식에 대한 자세한 정보를 찾을 수 있는 가장 효율적이고 효과적인 방법입니다.

IBM PartnerWorld for Developers 웹 사이트의 Application Factory 및 OS/400 PASE 페이지에는 어플리케이션을 OS/400 PASE에서 iSeries 서버로 이식하는 데 대한 일반 정보가 나와 있습니다. Application Factory는 어플리케이션을 iSeries 서버로 이식하기 위해 OS/400 PASE와 다른 솔루션을 비교합니다.

AIX 명령과 유틸리티에 대한 자세한 정보는 AIX 문서  웹 사이트를 참조하십시오.

Printed in U.S.A