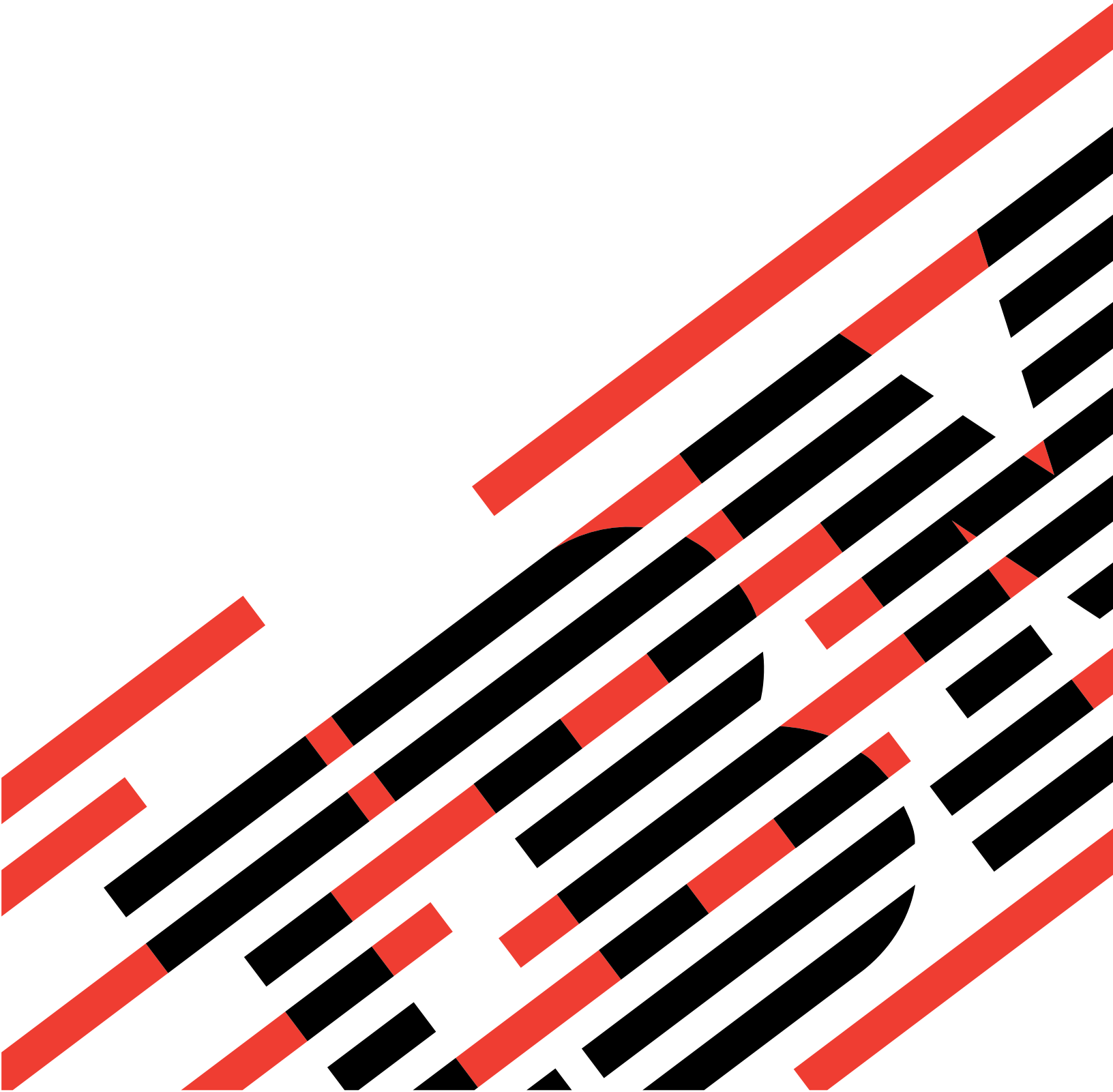
IBM

eserver

iSeries

OS/400 PASE

버전 5 릴리스 3



IBM

@server

iSeries

OS/400 PASE

버전 5 릴리스 3

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 59 페이지의 『주의사항』의 정보를 읽으십시오.

제 7 판(2005년 8월)

| 이 개정판은 새 개정판에 별도로 명시하지 않는 한, IBM Operating System/400(제품 번호 5722-SS1) 버전 5, 릴리스 3, 수정 0 | 및 모든 후속 릴리스와 수정에 적용됩니다. 이 버전은 모든 축약 명령어 세트 컴퓨터(RISC) 모델 및 CISC 모델에서 실행되지 않습 | 니다.

목차

OS/400 PASE

OS/400® PASE (OS/400 Portable Application Solutions Environment)르 사용하여 AIX® 어플리케이션을 최소한의 노력으로 iSeries™ 서버에 이식할 수 있습니다. OS/400 PASE는 복잡한 UNIX® 시스템 관리 없이 선택된 UNIX 어플리케이션을 실행할 수 있는 통합 런타임 환경을 제공합니다. 또한 OS/400 PASE는 강력한 스크립팅 환경을 제공하는 산업 표준 및 사실상의 표준 쉘과 유틸리티를 제공합니다.

OS/400 PASE에 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)에 대한 개요 및 OS/400 PASE가 도움이 되는 방법과 시기를 설명하고, OS/400 PASE를 iSeries 서버에 설치하는 지침을 제공합니다.

OS/400 PASE 계획
OS/400 PASE를 사용하기 전에 고려해야 할 몇 가지 기술적인 요구사항에 대해 설명합니다.

OS/400 PASE에서 실행할 프로그램 준비
OS/400 PASE에서 효과적으로 실행할 AIX 프로그램을 작성, 컴파일 및 복사하는 지침을 제공합니다.

OS/400 환경에서 OS/400 PASE 프로그램 사용
OS/400 환경에서 OS/400 PASE를 실행하는 방법, OS/400 PASE 프로그램에서 OS/400 프로그램과 ILE 프로시듀어를 호출하는 방법 및 OS/400 PASE 프로그램이 OS/400 기능(예: 보안, 메세징, 데이터베이스, 통신, 작업 관리, 인쇄) 및 ILE와 대화하는 방법을 설명합니다.

OS/400 PASE 디버그
어플리케이션의 문제를 식별하고 수정하는 데 사용할 수 있는 디버깅 툴에 대한 정보를 제공합니다.

성능 최적화
어플리케이션을 보다 효과적으로 실행할 수 있는 방법에 대한 정보를 제공합니다.

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

OS/400 PASE 관련 정보
OS/400 PASE API, 라이브러리 및 유틸리티에 대한 자세한 정보가 있는 iSeries Information Center의 위치를 표시합니다. OS/400 PASE 및 AIX에 대한 Information Center 외부의 추가 정보에 대한 링크를 제공합니다.

V5R3의 새로운 사항

이 페이지에서는 V5R3용 OS/400 PASE 제품의 변경사항을 설명합니다.

- V5R3M0용 OS/400 PASE는 AIX 5.2(OS/400용 AIX 5.1 PASE V5R1M0과 비교)에서 파생됩니다.
- 다음 컴파일러 제품은 OS/400 PASE(버전 V5R2M0 및 V5R3M0용)에서의 실행을 지원합니다.
  - AIX용 IBM® VisualAge® C++ Professional, 버전 6.0
  - AIX용 IBM C, 버전 6.0
  - AIX용 IBM XL Fortran, 버전 8.1.1
- OS/400 옵션 13(System Openness Includes) 또한 OS/400 PASE C와 C++ 프로그램을 컴파일하는 데 필요한 헤더 파일 지원(V5R2 및 V5R3)을 추가했습니다
- 다음 유틸리티는 신규 또는 변경된 유틸리티입니다.
  - df(파일 시스템의 공간에 대한 보고서 정보)
  - idlj(IDL 대 Java 컴파일러를 실행하도록 QShell `idlj` 명령 실행)
  - orbd(Java™ Object Request Broker 디먼을 실행하도록 QShell `orbd` 명령 실행)
  - servertool(Java IDL Server Tool을 실행하도록 QShell `servertool` 명령 실행)
  - OS/400 QP0ZCALL API 및 CL CALL 명령의 변경을 통해 OS/400 PASE 쉘 스크립트 qsh, qsh_inout 또는 qsh_out로 호출된 ILE 또는 OPM 유틸리티 프로그램에 전달할 수 있는 인수의 수를 증가시킵니다
- 다음과 같은 라이브러리가 추가되었습니다.
  - libnsl.a: 독립 리모트 프로시듀어 호출 전송(TI-RPC)
  - libtli.a: 라이브러리 인터페이스 전송
  - libxti.a: X/OPEN(TM) 라이브러리 인터페이스 전송
- 인수가 유효하지 않으면(예: 경로가 없거나 권한이 없거나 유효한 실행 가능 오브젝트가 아님) Qp2RunPase API, QP2SHELL 및 QP2TERM 프로그램에서 고유한 오류 메세지가 송신됩니다.
- OS/400 PASE 로더 구현이 64비트 AIX 커널에서 파생되어 64비트 공유 라이브러리 언로드와 같은 기능에 대한 향상된 성능을 제공하고 지원을 추가합니다. 64비트 커널을 반영하도록 런타임 인터페이스(예: `sysconf` 및 `_system_configuration`)가 갱신됩니다.
- 신규 또는 변경된 OS/400 PASE 런타임 함수:
  - _GETTS64(OS/400 PASE 주소에 대한 64비트 Teraspace 주소 가져오기)
  - _GETTS64_SPP(공백 포인터에서 64비트 Teraspace 주소 가져오기)
  - _GETTS64M(복수 64비트 Teraspace 주소 가져오기)
  - _ILECALLX가 ARG_MEMTS64 및 ARG_TS64PTR 지원을 추가함
  - _ILELOADX(64비트 ILE 활성 마크 지원)
  - _ILESYMX(64비트 ILE 활성 마크 지원)
  - 추가 인수에 대한 _PGMCALL 지원 및 ASCII 대 EBCDIC 스트링 변환 지원

  - _SETSPP_TS64(64비트 Teraspace 주소로 공백 포인터 설정)
  - _SETSPPM(복수 공백 포인터 설정)
  - PASE_FORK_JOBNAME 환경 변수에 대한 포크(fork) 및 f_fork 지원
  - fork400 및 f_fork400(작업명 및 자원 ID 지정)
  - getgrent(그룹 항목 가져오기)
  - getrpid(실제 프로세스 ID 가져오기)
  - getpwent(암호/사용자 항목 가져오기)
  - mntctl(마운트된 파일 시스템의 속성 검색)
  - Qp2setenv_ile(ILE 환경 변수 설정)
- 신규 또는 변경된 (ILE) OS/400 PASE용 API:
  - QP2SHELL, QP2SHELL2 및 QP2TERM 프로그램은 OS/400 TIMZON 작업 속성(시간대 지원)을 일치시키기 위해 PASE_TZ 환경 변수로 기본 설정됩니다.
  - QP2SHELL, QP2SHELL2 및 Qp2RunPase는 ILE 환경 변수 QIBM_PASE_FLUSH_STDIO를 검사하여 통합 파일 시스템 설명자를 사용하지 않을 때(QIBM_USE_DESCRIPTOR_STDIO가 설정되지 않은 경우) 표준 출력(stdout 및 stderr)을 한꺼번에 내보낼지 여부를 판별합니다.
  - 명명된 특정 기호를 임의 자료로 채우는 Qp2RunPase API에 대한 지원이 삭제되었습니다. Qp2RunPase(symbolName)에 대한 두 번째 인수는 널(null) 포인터이어야 합니다. OS/400 PASE 프로그램은 by-address 인수로 ILE 및 OPM 코드(_ILECALL 및 _PGMCALL 사용)를 호출하여 널(null) 종료 인수 또는 환경 변수 문자 스트링으로 표시할 수 없는 입력을 검색할 수 있습니다.
- 작업 시작 메세지가 포크(fork) 작업에 대한 작업 기록부나 QHST에 기록되지 않습니다. 작업 완료 메세지는 작업이 비정상적으로 종료한 경우에만 작성됩니다.
- 현재 OS/400 PASE 런타임이 UTC(Coordinated Universal Time)를 추적하는 시스템 시계를 사용하므로 변경 후 Qp2RunPase API가 처음 실행될 때까지 지연되지 않고 시간대 변경이 즉시 반영됩니다.
- OS/400 PASE는 이전에 UCS-2로 제한된 내부 시스템 지원 인터페이스(예: 파일 시스템)에 대한 UTF-16 코드화를 지원합니다.
- 각각의 UNIX 표준에 대해 *zombie* 프로세스 지원이 추가되었습니다.
- 새로운 로케일이 추가되었습니다.

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 정보의 PDF 버전은 신규 또는 변경된 정보를 여백을 따라 세로 막대로 표시합니다.

이 릴리스에 대한 새로운 사항이나 변경사항에 대한 다른 정보는 사용자 메모를 참조하십시오.

이 주제 인쇄

이 문서의 PDF 버전을 보거나 다운로드하려면 OS/400 PASE(약 282KB)를 선택하십시오.

PDF 파일 저장

PDF를 보거나 인쇄를 위해 워크스테이션에 저장하려면 다음을 수행하십시오.

- 브라우저에서 PDF를 마우스 오른쪽 단추로 클릭하십시오(위의 링크를 마우스 오른쪽 단추로 클릭).
- Internet Explorer를 사용하는 경우 **다른 이름으로 대상 저장...**을 클릭하십시오. Netscape Communicator를 사용하는 경우 **다른 이름으로 링크 저장...**을 클릭하십시오.
- PDF를 저장하려는 디렉토리로 가십시오.
- **저장**을 클릭하십시오.

Adobe Acrobat Reader 다운로드

해당 PDF를 보거나 인쇄하려면 Adobe Acrobat Reader가 필요합니다. Adobe 웹 사이트 (www.adobe.com/products/acrobat/readstep.html)  에서 사본을 다운로드할 수 있습니다.

OS/400 PASE 시작하기

교차 플랫폼 어플리케이션의 개발과 전개는 효율적 비즈니스 컴퓨팅 환경의 중요 구성요소입니다. 시스템에서 제공하는 함수를 쉽게 사용하고 통합하는 것도 중요합니다. 이 점이 iSeries 및 AS/400e™ 서버의 대표적인 특징이기도 합니다. 비즈니스가 점차 개방형 컴퓨팅 환경으로 이동해감에 따라 서로 상이한 이러한 목표를 달성하는 것이 어렵고, 시간 소모가 많으며 고비용의 작업이라는 것을 알게 될 것입니다. 예를 들어, AIX 오퍼레이팅 시스템에서 실행되고 해당 기능을 사용하는 익숙한 어플리케이션에서 이익을 얻기를 원하면서도 부가되는 AIX 및 OS/400 오퍼레이팅 시스템의 추가된 관리 부담은 원하지 않을 수 있습니다.

이런 경우에는 OS/400 PASE(OS/400 Portable Application Solution Environment)가 도움이 됩니다. OS/400 PASE를 사용하면 거의 변경 없이 OS/400에서 AIX 어플리케이션 2진의 대부분을 실행할 수 있으며 플랫폼 솔루션 포트폴리오를 효율적으로 확장시킬 수 있습니다.

OS/400 PASE에 대해 자세히 학습하려면 다음 주제를 참조하십시오.

OS/400 PASE 개념

어플리케이션 개발에서 OS/400 PASE가 유용한 경우

OS/400 PASE 설치

OS/400 PASE 개념

OS/400 PASE(OS/400 Portable Application Solutions Environment)는 OS/400에서 실행하는 AIX 어플리케이션에 대한 통합 런타임 환경입니다. AIX의 ABI(Application Binary Interface)를 지원하고 AIX 공유 라이브러리, 쉘 및 유틸리티가 제공하는 다양한 지원의 서브세트를 제공합니다. OS/400 PASE가 PowerPC® 기계 명령어의 직접 처리를 지원하므로 기계 명령어만 에뮬레이트하는 환경이 갖는 단점은 없습니다.

OS/400 PASE 어플리케이션

- C, C++, Fortran 또는 PowerPC 어셈블러에 기록할 수 있음
- AIX PowerPC 어플리케이션과 동일한 2진 실행 형식 사용
- OS/400 작업에서 실행
- 파일 시스템, 보안, 소켓 등과 같은 OS/400 시스템 기능 사용

OS/400 PASE는 OS/400의 UNIX 오퍼레이팅 시스템이 아닙니다. OS/400 PASE는 OS/400에서 거의 변경 없이 AIX 프로그램을 실행하도록 설계되었습니다. 기타 UNIX 기반 환경의 프로그램은 OS/400 PASE에서 실행하기 위한 첫 번째 단계로서 AIX에서 컴파일될 수 있도록 기록되어야 합니다.

OS/400 PASE 통합 런타임은 iSeries 서버의 사용권 내부 코드 커널에서 실행됩니다. 시스템은 OS/400 PASE와 다른 런타임 환경(ILE 및 Java 포함)에서 여러 가지 공통 OS/400 기능을 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 강화합니다.

최소한의 노력으로 빠른 어플리케이션 전개

대부분의 경우 AIX 프로그램은 거의 변경 없이 OS/400 PASE에서 실행될 수 있습니다. 필요한 AIX 프로그래밍 기술의 레벨은 AIX 프로그램의 설계에 따라 다릅니다. 또한 프로그램 설계에 추가 OS/400 어플리케이션 통합(예: CL 명령으로)을 제공하여 구성에 대한 어플리케이션 사용자의 걱정을 최소화할 수 있습니다.

OS/400 PASE는 OS/400의 성공을 공유할 솔루션 개발자를 위한 다른 이식 옵션을 추가합니다. 이식 시간을 크게 단축시키는 수단을 제공함으로써 OS/400 PASE는 시장 출시 시간과 솔루션 개발자의 투자 이익을 향상시킬 수 있습니다.

OS/400의 다양한 AIX 기술 서브세트

OS/400 PASE는 다음을 포함하여 AIX 기술의 다양한 서브세트를 기반으로 하는 어플리케이션 런타임을 구현합니다.

- 표준 C 및 C++ 런타임(스레드세이프 및 비스레드세이프)
- Fortran 런타임(스레드세이프 및 비스레드세이프)
- `pthreads` 스레드 패키지
- 자료 변환을 위한 `iconv` 서비스
- BSD(Berkeley Software Distributions)에 상당하는 지원
- Motif widget 세트와 함께 X Window System 클라이언트 지원
- 유사 단말기(PTY) 지원

어플리케이션은 OS/400 PASE가 지원하는 레벨과 호환되는 AIX의 레벨을 실행하는 AIX 워크스테이션에서 개발되고 컴파일된 다음 OS/400에서 실행됩니다.

또는 지원되는 컴파일러 제품 중 하나를 OS/400 PASE 환경에 설치하여 OS/400 PASE에서 어플리케이션을 개발, 컴파일, 빌드 및 실행할 수 있습니다. 자세한 정보는 AIX 소스 컴파일을 참조하십시오.

OS/400 PASE에는 또한 강력한 스크립팅 환경을 제공하는 Korn, Bourne 및 C 쉘과 약 200개의 유틸리티도 포함됩니다. 자세한 정보는 OS/400 PASE 쉘 및 유틸리티를 참조하십시오.

OS/400 PASE는 AIX 및 OS/400 오퍼레이팅 시스템에 대해 IBM의 일반 프로세서 기술을 사용합니다. PowerPC 프로세서는 OS/400 PASE 런타임에서 어플리케이션을 실행하도록 OS/400 모드에서 AIX 모드로 전환됩니다.

OS/400 PASE에서 실행되는 어플리케이션은 OS/400 통합 파일 시스템과 iSeries용 DB2 Universal Database™와 통합됩니다. 어플리케이션은 Java 및 ILE(Integrated Language Environment®) 어플리케이션을 호출할 수 있습니다(이들 어플리케이션에 의해 호출될 수도 있음). 일반적으로 보안, 메세지 처리, 통신, 백업 및 회복 등과 같은 OS/400 조작 환경의 모든 측면을 활용할 수 있습니다. 동시에 AIX 인터페이스에서 파생된 어플리케이션 인터페이스도 활용할 수 있습니다.

어플리케이션 개발에서 OS/400 PASE가 유용한 옵션이 되는 경우

OS/400 PASE는 AIX 어플리케이션을 iSeries 서버에 이식하는 방법을 결정하는 방법에 있어서 상당한 유연성을 제공합니다. 물론 OS/400 PASE는 선택할 수 있는 여러 옵션 중 하나입니다.

API 분석

어플리케이션이 OS/400 PASE에 적합한지 여부를 판별하는 출발점은 어플리케이션에 사용되는 API, 라이브러리, 유틸리티 및 OS/400에서 어플리케이션을 효과적으로 실행하는 방법 및 그 어플리케이션을 분석하는 것입니다. IBM PartnerWorld® 팀은 어플리케이션을 분석하고 잠재적인 장애물을 설명하는 무료 이식 평가 툴인 API 분석 툴을 제공합니다. 어플리케이션을 OS/400 PASE에 이식하는 프로시듀어에 분석 툴을 맞추는 방법에 대한 자세한 정보는 OS/400 PASE에서 실행할 프로그램 준비를 참조하십시오.

잠재적 OS/400 PASE 어플리케이션의 특성

다음은 OS/400 PASE의 사용 여부를 결정할 때 고려할 수 있는 몇 가지 유용한 지침입니다.

- **AIX 어플리케이션이 계산 집약적입니까?** OS/400 PASE는 최적화된 연산 라이브러리로 제공하여 iSeries 서버에서 계산 집약적 어플리케이션을 실행하는 좋은 환경을 제공합니다.
- **어플리케이션이 `fork()`, X Window System 또는 유사 단말기(PTY) 지원과 같이 OS/400 PASE에만 지원되는(또는 ILE에서 부분적으로만 지원됨) 기능에 의존하는 비중이 높습니까?** OS/400 PASE는 현재 OS/400 시스템에 없는 `fork()` 및 `exec()`를 지원합니다(`fork()` 함수를 `exec()` 함수와 통합하는 `spawn()` 제외).
- **어플리케이션이 복잡한 AIX 기반 빌드 프로세스 또는 테스트 환경을 사용합니까?** OS/400 PASE를 사용하면 새 플랫폼으로 쉽게 전송되지 않는 기존의 복잡한 프로세스가 있는 경우에 특히 유용한 AIX 기반 빌드 프로세스를 사용할 수 있습니다.

- **어플리케이션에 ASCII 문자 세트에 대한 종속성이 있습니까?** OS/400 PASE는 이러한 요구에 따라 어플리케이션에 좋은 지원을 제공합니다.
- **어플리케이션이 많은 포인터 조작을 수행하거나 정수를 포인터로 변환(캐스트)합니까?** OS/400 PASE는 최소의 수행 비용과 정수를 포인터로 변환하는 기능으로 32비트 및 64비트 AIX 주소지정 모델을 지원합니다.

OS/400 PASE가 최적의 솔루션이 될 수 없는 경우

일반적으로 OS/400 PASE는 ILE에서 호출해야 할 여러 호출 가능한 인터페이스를 제공하고 다음과 같은 특성을 갖는 코드에는 적합하지 않습니다.

- 호출할 때마다 OS/400 PASE를 시작 또는 종료하거나, 이미 사용 중인 OS/400 PASE 프로그램에서 Qp2CallPase API를 통해 OS/400 PASE 프로시듀어를 호출함으로써 제공된 것보다 더 높은 성능 호출 및 리턴을 필요로 하는 코드.
- ILE 호출자와 라이브러리 코드 사이에서 메모리나 이름공간을 공유해야 하는 코드 OS/400 PASE 프로그램은 호출된 ILE 코드와 이름공간 또는 메모리를 내재적으로 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. 향상된 DNS(Domain Name Server) 서버와 같은 일부 시스템 소프트웨어 및 ILE C++ 컴파일러에는 OS/400 PASE 지원이 필요합니다.

OS/400 PASE를 서버에 설치하려면 다음을 수행하십시오.

1. OS/400 명령행에서 `GO LICPGM`을 입력하십시오.
2. 11을 선택하고, 사용권 프로그램 설치를 선택하십시오.
3. 옵션 33(5722SS1 - Portable Application Solutions Environment)을 선택하십시오.
4. (선택사항) 추가 로케일을 설치하십시오. OS/400 PASE 제품은 OS/400에 설치한 언어 피처와 연관된 로케일 오브젝트만 설치합니다. 서버에서 언어 피처에 포함되지 않은 로케일이 필요한 경우 추가 OS/400 언어 피처를 주문하고 설치해야 합니다. 자세한 정보는 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/400과 함께 제공된 모든 라이브러리 코드를 사용할 수 있는 권한을 부여합니다. 그러나 이 사용권은 OS/400 PASE와 함께 제공되지 않은 AIX 라이브러리의 사용권을 의미하지는 않습니다. 모든 AIX 제품은 IBM에서 별도로 사용권을 부여합니다.

사용자 고유 어플리케이션을 OS/400 PASE에 이식하면 어플리케이션에는 OS/400 PASE와 함께 제공되지 않은 AIX 라이브러리에 대한 종속성이 있음을 알 수 있습니다. 이러한 라이브러리를 OS/400 시스템에 이식하려면 먼저 해당 라이브러리를 제공한 소프트웨어 제품을 판별하고 해당 소프트웨어 제품의 사용권 계약 조건

을 검토해야 합니다. 추가 미들웨어 종속성을 OS/400 시스템에 이식하려면 IBM 또는 제3자와의 협력이 필요합니다. 이식을 시작하기 전에 이식 중인 코드와 관련한 모든 사용권 계약을 면밀히 검토해야 합니다. IBM에 속한 라이브러리에 대한 사용권 계약을 찾으려면 IBM 영업대표, IBM Porting Center 중 하나, Rochester의 고객 기술 센터 또는 개발자용 PartnerWorld에 문의하십시오.

OS/400 PASE 계획

OS/400 PASE는 최소한의 노력으로 AIX 어플리케이션을 iSeries 서버에 이식할 수 있는 AIX 런타임 환경을 OS/400에 제공합니다. 실제로 많은 AIX 프로그램이 변경 없이 OS/400 PASE에서 실행됩니다. 그 이유는 OS/400 PASE가 AIX에서 사용할 수 있는 여러 동일한 공유 라이브러리를 제공하고, pSeries®AIXPowerPC 프로세서에서 실행되는 방식과 동일한 방식으로 iSeries PowerPC 프로세서에서 직접 실행되는 AIX 유틸리티의 다양한 서브세트를 제공하기 때문입니다.

OS/400 PASE에 대한 작업을 시작할 때 다음과 같은 사항에 유의하십시오.

- **OS/400 2진의 목표 릴리스와 2진이 실행되는 OS/400 PASE의 릴리스 사이에 상관이 있습니다.**

  AIX에서 OS/400 PASE 어플리케이션을 컴파일하는 경우 AIX에서 작성된 어플리케이션 2진이 어플리케이션을 실행할 OS/400 PASE의 버전과 호환되어야 합니다. 다음 표는 OS/400 PASE의 서로 다른 버전과 호환되는 AIX 2진 버전을 보여줍니다. 예를 들어, AIX 릴리스 5.1용으로 작성한 32비트 어플리케이션은 OS/400 PASE V5R3 또는 V5R2에서 실행되지만 OS/400 PASE V5R1 또는 V4R5에서는 실행되지 않습니다. 마찬가지로, AIX 릴리스 4.3용으로 작성한 64비트 어플리케이션은 OS/400 PASE V5R1에서 실행되지만 OS/400 PASE V5R3, V5R2 또는 V4R5에서는 실행되지 않습니다.

AIX 릴리스	OS/400 V4R5	OS/400 V5R1	OS/400 V5R2	OS/400 V5R3
4.2(32비트)	X	X	X	X
4.3(32비트)	X	X	X	X
4.3(64비트)	-	X	-	-
5.1(32비트 또는 64비트)	-	-	X	X
5.2(32비트 또는 64비트)	-	-	-	X

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

  대신에 공유 라이브러리에 필요한 하위 시스템 함수는 OS/400 커널 또는 통합 OS/400 함수로 라우트됩니다. 그러므로 OS/400 PASE는 AIX와 OS/400 플랫폼 사이의 간격을 메워줍니다. 코드는 공유 라이브러리의 API에 AIX에서와 같은 구문을 사용하지만 OS/400 PASE 프로그램은 OS/400 작업에서 실행되고 기타 OS/400 작업처럼 OS/400에 의해 관리됩니다.

- **대부분의 경우 OS/400 PASE에서 호출하는 API는 AIX에서와 동일한 방식으로 작동합니다.**

  그러나 일부 API는 OS/400 PASE에서 다르게 작동될 수 있거나, OS/400 PASE에서 지원되지 않을 수도 있습니다. 이런 이유로 인해 OS/400 PASE 프로그램 준비 계획은 먼저 API 분석 툴을 사용한 철저한 분석을 통해 시작되어야 합니다. 이 툴은 AIX 어플리케이션을 OS/400 PASE에 이식할 때 고려해야 할 프로그램 수정 유형에 대한 포괄적인 요약 사항을 제공합니다.

- **AIX와 OS/400 플랫폼 간의 일부 차이점을 고려하십시오.**
  - AIX는 대소문자를 구분하지만 특정 OS/400 파일 시스템은 대소문자를 구분하지 않습니다.
  - AIX는 자료 코드화에 ASCII를 사용하지만 OS/400은 EBCDIC를 사용합니다. 이는 OS/400 PASE 프로그램에서 ILE(Integrated Language Environment) 코드 호출에 대한 세부사항을 관리할 경우에 고려해야 합니다. 예를 들어 OS/400 PASE에서 임의의 ILE 프로시듀어로 호출할 때 스트링의 문자 코드화 변환을 처리하도록 OS/400 PASE 프로그램을 명시적으로 코딩해야 합니다. OS/400 PASE 런타임 지원은 문자 코드화 변환을 위한 `iconv_open()`, `iconv()` 및 `iconv_close()` 함수를 포함합니다.

    주: OS/400 PASE와 ILE는 각각 자체의 변환표를 사용하여 `iconv()` 인터페이스를 독립적으로 구현합니다. OS/400 PASE `iconv()` 지원에서 지원하는 변환은 통합 파일 시스템에 바이트 스트림 파일로 저장되므로 사용자가 수정하고 확장할 수 있습니다.
  - AIX 어플리케이션에서는 행(예: 파일과 쉘 스크립트의 행)이 라인 피드(LF)로 끝나지만 퍼스널 컴퓨터(PC) 소프트웨어인 OS/400 소프트웨어에서는 행이 캐리지 리턴 및 라인 피드(CRLF)로 끝나야 합니다.
  - AIX에서 사용할 스크립트와 프로그램은 표준 유틸리티에 대한 하드 코딩된 경로를 사용할 수 있으므로 OS/400 PASE에서 사용할 경로가 적용되도록 경로를 수정해야 합니다. 자세한 정보는 OS/400 PASE와 프로그램의 호환성 분석을 참조하십시오.

  OS/400 PASE는 자동으로 이러한 문제를 처리합니다. 예를 들어, 시스템에서 제공하는 OS/400 PASE 런타임 서비스(OS/400 옵션 33과 함께 제공된 공유 라이브러리의 시스템 호출 또는 런타임 함수 포함)를 사용할 경우 파일 설명자(바이트 스트림 파일 또는 소켓)에 쓰거나 읽은 자료에 대한 변환이 수행되지 않더라도 OS/400 PASE는 필요한 경우 ASCII-EBCDIC 간의 변환을 수행합니다.

  다른 하위 함수(예: _ILECALL)를 사용하여 ILE 함수와 API로의 호출에 OS/400 PASE 프로그램의 기능을 확장할 수 있지만 위에서 언급한 것처럼 자료 변환 처리가 필요할 수 있습니다. 또한 프로그램에서 이와 같은 확장을 코딩하려면 추가적으로 헤더와 내보내기 파일을 사용해야 합니다.

OS/400 PASE에서 실행할 프로그램 준비

OS/400에서 효과적으로 실행되는 AIX 프로그램 준비를 위해 수행할 단계는 프로그램의 특성 및 OS/400 고유 인터페이스와 함수의 사용 여부에 따라 다릅니다.

UNIX 어플리케이션을 OS/400 PASE에 이식하려면 먼저 어플리케이션이 AIX 컴파일러를 사용하여 컴파일하는지를 확인해야 합니다. 이 요구사항에 맞게 UNIX 프로그램을 수정해야 되는 경우도 있습니다.

OS/400 PASE에 대한 프로그램을 준비하는 경우

다음 주제를 참조하여 OS/400 PASE에서 사용할 수 있는 프로그램을 준비하십시오.

프로그램 분석

이 프로세스의 첫 단계는 모든 경우에 수행이 권장됩니다. API 분석 툴을 사용하여 프로그램에서 사용하는 API 및 OS/400 PASE에서 API를 수행할 수 있는 방법에 대한 자세한 보고서를 얻으십시오.

AIX 소스 프로그램 컴파일
프로그램이 OS/400 PASE 프로그램으로서 적합한지 판별하고 OS/400 PASE에서 실행할 수 있도록 수정한 후에 소스를 컴파일하십시오. AIX 프로그램 분석 결과 OS/400 PASE에서 실행하기 위해 변경이 필요하지 않은 경우 프로그램을 다시 컴파일할 필요가 없습니다.

AIX 시스템을 사용하여 OS/400 PASE 프로그램을 컴파일하거나, 선택적으로 지원되는 AIX 컴파일러 제품 중 하나를 OS/400 PASE에 설치하여 OS/400 PASE 환경에서 프로그램을 컴파일할 수도 있습니다.

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

AIX 어플리케이션을 사용자 정의하여 OS/400 인터페이스 사용(선택사항)
AIX 어플리케이션을 사용자 정의하여 OS/400 고유 인터페이스를 사용하고 AIX에서 어플리케이션을 컴파일하는 경우에는 OS/400 PASE 프로그램을 컴파일하기 전에 OS/400 헤더 또는 내보내기 파일을 AIX 시스템에 복사해야 합니다.

프로그램과 OS/400 PASE의 호환성 분석

iSeries 서버에 대한 UNIX C 어플리케이션의 이식성을 평가하는 첫 번째 단계는 어플리케이션에 사용된 인터페이스를 분석하는 것입니다. 이 API 분석에서는 산업 표준이 아니며 OS/400에서 지원되지 않는 어플리케이션에서 사용된 해당 인터페이스를 식별합니다. 또한 UNIX 시스템과 OS/400 구조가 다르기 때문에 표준과는 호환되지만 서로 다르게 지원되는 인터페이스를 식별합니다.

API 분석 툴  은 프런트엔드 프로세스 및 백엔드 프로세스로 구성됩니다. 프런트엔드 프로세스는 컴파일된 어플리케이션을 스캔하여 어플리케이션에서 사용하는 인터페이스(외부 기능 및 자료)를 추출하고 이들 모든 인터페이스의 리스트를 생성합니다. 백엔드 프로세스는 이 인터페이스 리스트를 입력하여 일반 시스템 API 및 이의 지원 데이터베이스와 이 인터페이스를 비교합니다.

API 분석 툴의 프런트엔드 프로세스는 UNIX 쉘 스크립트입니다. `nm` 또는 `dump` 명령을 사용하여 어플리케이션의 외부 기호 표에서 기호 정보를 찾습니다.

기호가 제거된 2진에는 툴이 분석을 수행하기에 충분한 동적 바인딩 정보가 들어 있습니다. 정적으로 바인드시킨 2진은 라이브러리 인터페이스를 분석에서 제거하지만 분석을 위해 시스템 호출 종속성은 보여줍니다.

컴파일하기 전에 수행할 추가 분석

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

- 어플리케이션에서 사용한 라이브러리 리스트 가져오기

  분석 툴은 어플리케이션에서 사용하는 일부 표준 API에 대한 피드백을 제공하지만 많은 공통 API 세트를 찾지는 못합니다. 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
  ```

- 코드에서 하드 코드 경로명 검사

  증명서를 변경하는 프로그램을 실행하거나 OS/400 PASE 환경 변수가 PASE_EXEC_QOPENSYS=N인 경우에도 프로그램 또는 스크립트를 실행하려면 하드 코드 경로명을 변경해야 합니다.

  /usr/bin/ksh가 절대 경로(루트에서 시작)이므로 이 경로가 없거나 bytestream 파일이 아니면 OS/400 PASE는 /QOpenSys 파일 시스템에서 /QOpenSys/usr/bin/ksh 경로명을 탐색합니다. QShell 유틸리티 프로그램은 bytestream 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

프로그램이 AIX 인터페이스만 사용하는 경우 필수 AIX 헤더를 사용하여 컴파일하고 AIX 라이브러리로 링크하여 OS/400 PASE의 2진을 준비하십시오. OS/400 PASE는 AIX 시스템 제공 공유 라이브러리와 정적으로 바인드된 어플리케이션을 지원하지 않습니다.

OS/400 PASE 프로그램 구조는 PowerPC용 AIX 프로그램과 구조적으로 동일합니다.

OS/400 PASE 옵션 33은 컴파일러를 포함하지 않습니다. AIX 시스템을 사용하여 OS/400 PASE 프로그램을 컴파일하거나 OS/400 PASE에서의 설치를 지원하는 AIX 컴파일러 제품 중 하나를 선택적으로 설치하여 OS/400 PASE 환경에서 사용자 프로그램을 컴파일할 수도 있습니다.

pSeries 서버에서 AIX 컴파일러 사용

PowerPC용 AIX ABI(Application Binary Interface)와 호환되는 출력을 생성하는 AIX 컴파일러 및 링크 프로그램을 사용하여 OS/400 PASE 프로그램을 빌드할 수 있습니다. OS/400 PASE는 PowerPC에 없는 POWER™ 구조 명령어(캐시 관리POWER 명령어 제외)를 사용하는 2진에 대한 명령어 에뮬레이션 지원을 제공합니다.

OS/400 PASE에서 AIX 컴파일러 사용

OS/400 PASE는 OS/400 PASE 환경에 다음과 같이 독립적으로 사용할 수 있는 AIX 컴파일러 설치를 지원합니다.

- AIX용 IBM VisualAge C++ Professional, 버전 6(5765-F56). (이 제품은 AIX용 IBM C 컴파일러를 포함합니다.)
- AIX용 IBM C, 버전 6(5765-F57)
- AIX용 IBM XL Fortran(5765-F70), 버전 8.1.1 이상

이 제품을 사용하여 iSeries 서버의 모든 OS/400 PASE 환경에서 사용자 AIX 어플리케이션을 개발, 컴파일, 빌드 및 실행할 수 있습니다.

이 제품의 확보 및 설치 방법에 대한 자세한 정보는 13 페이지의 『OS/400 PASE에 AIX 컴파일러 설치』를 참조하십시오.

개발 툴

AIX에서 사용할 여러 개발 툴(예: ld, ar, make, yacc)이 OS/400 PASE와 함께 포함됩니다. 자세한 내용은 OS/400 PASE 쉘 및 유틸리티 주제를 참조하십시오. 다른 소스의 여러 AIX 툴(예: 오픈 소스 툴 gcc)은 OS/400 PASE에서도 사용할 수 있습니다.

iSeries Tools for Developers PRPQ(5799-PTL)에도 iSeries 어플리케이션의 개발, 빌드 및 이식을 지원하는 다양한 툴 배열이 있습니다. 이 PRPQ에 대한 자세한 정보는 Porting - iSeries Tools for developers 웹 사이트를 참조하십시오.

포인터 처리를 위한 컴파일러 주

- 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 정렬되도록 하기 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 옵션을 가져와서 libc128.a로 링크하는 쉬운 방법은 xlc 명령 대신 xlc128 명령을 사용하는 것입니다.
- xlc/xlC 컴파일러는 현재 정적 변수나 자동 변수에 대해 16바이트 정렬을 강제하는 수단을 제공하지 않습니다. 단순히 이 컴파일러는 구조 내에서 128비트 long double 필드에 대한 상대적인 정렬을 보장합니다. OS/400 PASE malloc 버전이 항상 16바이트 정렬 기억장치를 제공하므로 스택 기억장치의 16바이트 정렬을 배열할 수 있습니다.
- 헤더 파일 as400_types.h는 유형 long long에서도 64비트 정수가 됩니다. xlc 컴파일러의 -qlonglong 옵션이 이 구조를 보장합니다(xlc 컴파일러를 실행하는 모든 명령의 디폴트가 아님).

예

다음 예는 AIX 시스템에서 OS/400 PASE 프로그램을 컴파일할 때 사용할 수 있습니다. OS/400 PASE에 설치된 컴파일러를 사용하여 사용자 프로그램을 컴파일하면 이 파일은 OS/400 시스템의 디폴트 경로 위치(예: /usr/include/ 및 /usr/lib/)에 있으므로 OS/400 고유 내보내기 또는 OS/400 고유 헤더 파일의 위치에 컴파일러 옵션을 지정하지 않아도 됩니다.

예 1: AIX 시스템의 다음 명령은 libc.a에서 내보낸 OS/400 고유 인터페이스를 사용할 수 있는 OS/400 PASE 프로그램 testpgm을 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 하기 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
```

이 예에서, iSeries용 DB2® UDB CLI(Call Level Interface)에 대한 OS/400 PASE 지원을 사용할 경우 빌드 명령에도 `-bI:/pase/include/libdb400.exp`를 지정해야 합니다.

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

OS/400 PASE에 AIX 컴파일러 설치

별도로 사용할 수 있는 다음 AIX 컴파일러 중 하나를 OS/400 PASE 환경에 설치할 수 있습니다.

- AIX용 IBM VisualAge C++ Professional, 버전 6(5765-F56). (이 제품은 AIX용 IBM C 컴파일러를 포함합니다.)
- AIX용 IBM C, 버전 6(5765-F57)
- AIX용 IBM XL Fortran(5765-F70), 버전 8.1.1 이상

이 제품을 사용하면 iSeries 서버의 OS/400 PASE 환경에서 전체적으로 AIX 어플리케이션을 개발, 컴파일, 빌드 및 실행할 수 있습니다. 이 제품의 주문 및 설치에 대한 정보는 Enablement roadmaps & resources

 웹 사이트를 참조하십시오. 웹 사이트에 문서화된 설치 지침은 다음과 같습니다.

AIX 컴파일러 설치

OS/400 PASE는 AIX 시스템에서 어플리케이션 설치 시 일반적으로 사용되는 AIX `smit` 또는 `installp` 유틸리티를 지원하지 않습니다. AIX용 VisualAge C++ Professional 또는 C 제품은 각 컴파일러의 설치 매체에 포함된 "비 디폴트 설치" 스크립트를 통해 수행됩니다.

다음 단계에서는 AIX용 VisualAge C++ Professional V6.0 또는 AIX용 C 제품을 iSeries OS/400 PASE에 설치합니다.

1. 필요한 전제조건이 있는지 확인하십시오. 컴파일러 설치 매체(AIX용 VisualAge C++ Professional의 5765-F56 또는 AIX용 C의 5765-F57) 이외에도, 컴파일러를 설치 및 사용하려면 iSeries 서버에 다음을 설치해야 합니다.
   - 5722SS1 옵션 33 - OS/400 PASE 자체

- 5722SS1 옵션 13 - 시스템 개방성 포함, /usr/include 통합 파일 시스템 디렉토리의 컴파일러 헤더 파일 포함
- Perl. 컴파일러 설치 스크립트에는 Perl이 필요합니다. 다음은 Perl을 설치하는 두 가지 방법입니다.
  - 5799PTL - iSeries Tools for Developers PRPQ. Perl(여러 가지 다른 개발 툴과 함께)은 별도로 사용할 수 있는 iSeries Tools For Developers PRPQ에 포함됩니다.
  - http://www.cpan.org/ports/#os400 - OS/400 PASE용 Perl 포트 2진 분배

2. 컴파일러 제품 설치 CD를 iSeries CDROM 장치에 삽입하십시오.
3. *ALLOBJ 권한이 있는 사용자 프로파일로 OS/400에 사인 온하십시오. 이 사용자 프로파일에서 컴파일러 제품 파일을 소유합니다.
4. 다음 CL 명령을 입력하여 대화식 OS/400 PASE 단말기 세션을 시작하십시오.

   ```
   call qp2term
   ```

5. 다음 명령을 입력하여 해당 컴파일러 설치 스크립트를 복원하십시오.
   - AIX용 VisualAge C++ Professional의 경우:

     ```
     cd / restore -qf /QOPT/CDROM/USR/SYS/INST.IMA/VACPP.NDI ./usr/vacpp/bin/vacppndi
     ```

   - AIX용 C의 경우:

     ```
     cd / restore -qf /QOPT/CDROM/USR/SYS/INST.IMA/VAC.NDI ./usr/vac/bin/vacndi
     ```

   - AIX용 XL Fortran의 경우:

     ```
     cd / restore -qf /QOPT/CDROM/USR/SYS/INST.IMA/XLF.NDI
     ```

6. 설치 스크립트를 실행하여 컴파일러를 설치하십시오. 컴파일러의 대상 디렉토리는 해당 명령의 -b 옵션에서 지정합니다. 컴파일러에 권장되는 디렉토리 이름은 다음 명령에 사용됩니다. 서로 다른 디렉토리를 선택할 경우에는 디렉토리가 /QOpenSys 트리에 있어야 합니다(대소문자 구분 파일명 허용).
   - AIX용 VisualAge C++ Professional의 경우(하나의 긴 명령으로 입력):

     ```
     /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vacpp/bin/vacppndi -i
       -d /QOPT/CDROM/USR/SYS/INST.IMA -b /QOpenSys/vac600
     ```

   - AIX용 C의 경우(다시 하나의 긴 명령으로 입력):

     ```
     /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vac/bin/vacndi -i
       -d /QOPT/CDROM/USR/SYS/INST.IMA -b /QOpenSys/vac600
     ```

   - AIX용 XL의 경우(다시 하나의 긴 명령으로 입력):

     ```
     /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/xlf/bin/xlfndi -i -d
     /QOPT/CDROM/USR/SYS/INST.IMA -b /QOpenSys/xlf811
     ```

7. OS/400 PASE에서 사용할 수 있도록 컴파일러가 설치되었습니다.

xlC와 같은 AIX용 VisualAge C++ Professional 컴파일러 명령은 /QOpenSys/vac600/usr/vacpp/bin/ 디렉토리에 있습니다. 이 디렉토리를 $PATH 환경 변수에 추가할 수 있습니다.

AIX용 VisualAge C++ Professional 컴파일러 문서는 /QOpenSys/vac600/usr/vacpp/pdf/en_US/ 디렉토리에 있으며 Adobe Acrobat 형식으로 되어 있습니다.

xlc 및 cc와 같은 AIX용 C 컴파일러 명령은 /QOpenSys/vac600/usr/vac/bin/ 디렉토리에서 찾을 수 있습니다. 이 디렉토리를 $PATH 환경 변수에 추가할 수 있습니다.

AIX용 C 컴파일러 문서는 /QOpenSys/vac600/usr/vac/pdf/en_US/ 디렉토리에 있으며 Adobe Acrobat 형식으로 되어 있습니다.

xlf와 같은 AIX용 XL Fortran 컴파일러 명령은 /QOpenSys/xlf811/usr/bin/ 디렉토리에 있습니다. 이 디렉토리를 $PATH 환경 변수에 추가할 수 있습니다.

AIX용 XL Fortran 컴파일러 문서는 /QOpenSys/xlf811/usr/share/man/info/en_US/xlf/pdf/ 디렉토리에 있으며 Adobe Acrobat 형식으로 되어 있습니다.

PTF 갱신 지침

AIX용 VisualAge C++ Professional 또는 C 제품에 대한 프로그램 임시 수정(PTF) 설치는 초기 컴파일러 설치에 사용된 것과 동일한 "비 디폴트 설치" 스크립트를 사용하여 수행됩니다. 다음 단계에서는 AIX용 VisualAge C++ Professional V6.0 PTF 또는 AIX용 C 제품을 iSeries OS/400 PASE에 설치합니다. PTF를 설치하려면 먼저 위에 설명된 단계를 사용하여 컴파일러를 설치해야 합니다.

1. 설치할 PTF 패키지 파일을 받으십시오. AIX VisualAge C++ 웹 사이트의 지원 다운로드 섹션에서 컴파일러 PTF 패키지의 압축된 TAR 이미지를 다운로드해야 합니다.
2. PTF 패키지 파일의 압축을 푸십시오. 압축된 TAR 이미지를 /QOpenSys/vacptf/ 디렉토리로 다운로드한 경우 QP2TERM 명령행에서 다음 명령을 사용하여 이를 수행할 수 있습니다.

   ```
   cd /QOpenSys/ptf
   uncompress <filename.tar.Z>
   tar -xvf <filename.tar>
   ```

3. 설치할 PTF 패키지의 리스트를 포함하는 파일을 작성하십시오. QP2TERM 명령행에서 다음 명령을 사용하여 이를 수행하십시오.

   ```
   cd /QOpenSys/ptf
   ls *.bff > ptflist.txt
   ```

4. 설치 스크립트를 실행하여 갱신 중인 컴파일러를 PTF를 기반으로 설치한 후 QP2TERM 명령행에서 다음 명령 중 하나를 입력하십시오.
   - AIX용 VisualAge C++ Professional의 경우(하나의 긴 명령으로 입력):

     ```
     /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vacpp/bin/vacppndi
     -d /QOpenSys/ptf -b /QOpenSys/vac600 -u /QOpenSys/ptf/ptflist.txt
     ```

   - AIX용 C의 경우(다시 하나의 긴 명령으로 입력):

     ```
     /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vac/bin/vacndi
     -d /QOpenSys/ptf -b /QOpenSys/vac600 -u /QOpenSys/ptf/ptflist.txt
     ```

   - AIX용 XL의 경우(다시 하나의 긴 명령으로 입력):

     ```
     /QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/xlf/bin/xlfndi
     -d /QOpenSys/ptf -b /QOpenSys/xlf811 -u /QOpenSys/ptf/ptflist.txt
     ```

설치 스크립트는 PTF 갱신 전의 컴파일러 파일의 압축된 TAR 백업을 작성합니다. 이 지침에서 표시한 대로 디렉토리를 사용하는 경우 이 파일은 /QOpenSys/vac600.backup.tar.Z로 명명됩니다. PTF 갱신 또는 PTF 갱신 자체를 설치할 때 문제가 발생하면 이 백업에서 복원하여 PTF 갱신을 설치제거하십시오.

OS/400 PASE 프로그램을 iSeries 서버에 복사

OS/400 PASE에서 실행하려는 AIX 2진을 통합 파일 시스템에 복사하십시오. 통합 파일 시스템에서 사용할 수 있는 모든 파일 시스템은 OS/400 PASE에서 사용할 수 있습니다. 통합 파일 시스템에 대한 자세한 정보는 통합 파일 시스템 주제를 참조하십시오.

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

- 대소문자 구분: 어플리케이션이 대소문자를 구분하려면 대소문자를 구분하면 작성된 사용자 정의 파일 시스템 또는 /QOpenSys 파일 시스템으로 이동하십시오.
- 행 종료 문자: AIX 및 OS/400은 텍스트 파일(예: 파일 및 쉘 스크립트)에서 서로 다른 행 종료 문자를 사용합니다.

파일 전송

다음 방법을 사용하여 OS/400 PASE 프로그램 및 관련 파일을 iSeries 서버에서 송수신할 수 있습니다.

- 파일 전송 프로토콜(FTP)을 사용하여 프로그램 복사
- 서버 메세지 블록(SMB)을 사용하여 프로그램 복사
- 리모트 파일 시스템을 사용하여 프로그램 복사

파일 전송 프로토콜(FTP)을 사용하여 프로그램 복사

OS/400 FTP 디먼 및 클라이언트를 사용하여 파일을 OS/400 통합 파일 시스템에서 송수신할 수 있습니다. 2진 모드로 파일을 전송하십시오. FTP 부속 명령 binary를 사용하여 이 모드를 설정하십시오.

파일을 통합 파일 시스템에 넣을 때 명명 형식 1(OS/400 FTP 명령의 NAMEFMT 1 부속 명령)을 사용해야 합니다. 이 형식은 UNIX 경로명 사용을 허용하고 스트림 파일에 해당 파일을 전송합니다. 명명 형식 1로 들어가려면 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 FTP 부속 명령 quote site namefmt 1을 사용하거나 namefmt 1을 로컬 클라이언트로 사용하십시오.

FTP에 대한 자세한 정보는 FTP 주제를 참조하십시오.

서버 메세지 블록(SMB)을 사용하여 프로그램 복사

OS/400은 SMB 클라이언트 및 서버 구성요소를 지원합니다. NetServer™가 구성되어 실행 중이면 OS/400 PASE는 /QNTC 파일 시스템을 통해 네트워크의 SMB 서버에 액세스합니다. UNIX 플랫폼에서는 동일한 서

비스를 제공하는 데 SAMBA 서버가 필요합니다. UNIX 시스템(예: AIX)을 설치 및 구성하면 작동시키면 OS/400 PASE에서 사용할 수 있는 디렉토리 및 파일을 작성할 수 있습니다.

리모트 파일 시스템을 사용하여 프로그램 복사

OS/400에서는 네트워크 파일 시스템(NFS)의 파일 시스템을 통합 파일 시스템 파일 공간의 마운트 위치에 마운트할 수 있습니다. AIX는 분산 파일 시스템(DFS™) 및 Andrew File System(AFS®)과 함께 NFS를 지원(DFS 대 NFS 및 AFS 대 NFS 변환 프로그램 사용)하기 때문에 이 파일 시스템은 OS/400에서 내보내고 마운트할 수 있습니다. 이를 통해 OS/400 PASE 어플리케이션은 이 파일 시스템을 사용할 수 있습니다. 액세스할 파일 또는 디렉토리 경로에 대한 보안 권한이 OS/400 사용자 프로파일의 사용자 ID 번호 및 그룹 ID 번호를 통해 검증됩니다. 여러 플랫폼에서 같은 사람으로 인식되는 사용자 프로파일은 모든 시스템에서 같은 사용자 ID를 가져야 합니다.

OS/400은 NFS 서버로 사용하는 것이 가장 효과적입니다. 이 경우 AIX 시스템에서 OS/400 통합 파일 시스템의 디렉토리로 마운트하고, AIX는 빌드할 때 프로그램을 OS/400에 직접 기록합니다.

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

대소문자 구분

주로 UNIX 시스템 인터페이스에서 대소문자를 구분합니다. OS/400에서 항상 대소문자를 구분하지는 않습니다. 대소문자 구분이 기존 코드를 복잡하게 만드는 경우 등 몇몇 상황에 특히 유의해야 합니다.

디렉토리 또는 파일에서 대소문자 구분은 OS/400에서 사용하는 파일 시스템에 따라 다릅니다. /QOpenSys 파일 시스템은 대소문자를 구분하므로 대소문자가 구분되는 사용자 정의 파일 시스템(UDFS)을 작성할 수 있습니다. 여러 파일 시스템의 특성에 대한 정보는 파일 시스템 비교 주제를 참조하십시오.

예

다음은 대소문자 구분으로 인해 발생할 수 있는 몇 가지 문제점입니다.

예 1: 이 예에서 쉘은 `readdir()`이 리턴하는 것에 대한 총칭명 접두부의 문자 비교를 수행합니다. 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` 유틸리티에서 사용자 이름의 대소문자를 구분하므로 -u 옵션에 지정한 대문자 이름과 OS/400 PASE 런타임 함수 `getpwuid()`에서 리턴된 소문자 이름이 일치하는지를 구분하지 않습니다.

```
$ 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 행(예: 파일과 쉘 스크립트의 행)이 라인 피드(LF)로 끝나야 합니다. 그러나 PC 소프트웨어와 일반 OS/400 소프트웨어에서는 행이 종종 캐리지 리턴 및 라인 피드(CRLF)로 끝납니다.

FTP에 사용된 CRLF

이러한 차이로 인해 문제가 발생할 수 있는 한 가지 예는 FTP를 사용하여 소스 파일과 쉘 스크립트를 AIX에서 iSeries로 변환하는 경우입니다. 표준 FTP는 텍스트 모드에서 송신된 자료를 호출하여 행의 끝에서 캐리지 리턴 및 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 나타난 사항을 정확하게 기록하고, CRLF를 텍스트 모드로 보유하여 OS/400 PASE 런타임과 유틸리티에 문제를 일으킵니다.

가능하면 UNIX 시스템에서 2진 모드 전송을 사용하여 이 문제를 방지하십시오. 대부분의 경우 퍼스널 컴퓨터에서 전송된 텍스트 파일은 CRLF 구분 행이 있습니다. 먼저 파일을 AIX로 전송하면 문제가 해결될 수 있습니다. 다음 조치는 현재 디렉토리에 있는 파일에서 CR을 제거합니다.

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

iSeries 및 PC 편집기에 사용된 CRLF

또한 iSeries 서버에서 편집기를 사용하거나 워크스테이션에서 편집기(예: Windows® 메모장 편집기)를 사용하여 파일 또는 쉘 스크립트를 편집할 경우에도 문제가 발생할 수 있습니다. 이러한 편집기는 OS/400 PASE가 의도한 LF를 사용하는 대신 CRLF를 새 행 분리자로 사용합니다.

CRLF를 새 행 분리자로 사용하지 않는 여러 편집기(예: ez 편집기)를 사용할 수 있습니다. 여러 개발자용 iSeries 툴  의 리스트를 참조하십시오.

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

AIX 어플리케이션에서 시스템 제공 OS/400 PASE 공유 라이브러리가 직접 지원하지 않는 OS/400 기능을 사용하려면 어플리케이션 준비를 위한 추가 단계를 수행해야 합니다.

1. OS/400 고유 기능에 대한 액세스를 조정하는 필수 OS/400 PASE 런타임 함수를 호출하도록 AIX 어플리케이션을 코딩하십시오.
2. AIX 시스템에서 OS/400 PASE 프로그램을 컴파일하는 경우 사용자 정의된 어플리케이션을 컴파일하기 전에 다음 단계를 수행하십시오.
   a. AIX 시스템에 필수 OS/400 고유 헤더 파일 복사

b. AIX 시스템에 필수 OS/400 고유 내보내기 파일 복사

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

27 페이지의『OS/400 PASE 프로그램에서 OS/400 프로그램 및 프로시듀어 호출』

39 페이지의『OS/400 PASE 프로그램과 OS/400의 대화 방법』

헤더 파일 복사

OS/400 PASE는 OS/400 고유 지원을 위해 헤더 파일로 표준 AIX 런타임을 증대시킵니다. 이 헤더 파일은 OS/400 PASE 및 OS/400 오퍼레이팅 시스템에서 제공합니다.

iSeries 서버에서 헤더 파일 탐색 경로의 AIX 기계로 헤더 파일을 복사하십시오.

헤더 파일을 다음 AIX 디렉토리 또는 컴파일러용 헤더 파일 탐색 경로에 있는 기타 디렉토리로 복사할 수 있습니다.

```
/usr/include
```

/usr/include가 아닌 디렉토리를 사용할 경우 AIX 컴파일러 명령에 -I 옵션을 사용하여 헤더 파일 탐색 경로에 이 디렉토리를 추가할 수 있습니다.

파일 복사에 대한 자세한 정보는 iSeries 서버로 OS/400 PASE 프로그램 복사를 참조하십시오.

OS/400 PASE 헤더 파일 복사

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

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

OS/400 PASE는 다음과 같은 헤더 파일을 제공합니다.

as400_protos.h	OS/400 PASE 대 ILE. 기타 OS/400 고유 함수를 제공합니다.
as400_types.h	ILE를 호출할 수 있는 고유한 OS/400 매개변수 유형 이 헤더 파일은 16바이트 기계 인터페이스(MI) 포인터에 대해 128비트 필드가 되는 long double 유형을 사용하는 ILEpointer 유형을 선언합니다. as400_types.h에 선언된 기타 유형은 유형 long long에서 64비트 정수가 됩니다. as400_types.h에 적절한 크기 및 정렬 유형이 선언되었는지 확인하려면 AIX 컴파일러를 -qlngdbl128, -qalign=natural 및 -qlonglong 옵션으로 실행해야 합니다.
os400msg.h	OS/400 메세지를 송수신할 함수

OS/400 헤더 파일 복사

OS/400 PASE 어플리케이션의 다른 OS/400 함수에 액세스하려면 사용 중인 OS/400 함수의 헤더 파일을 개발 기계에 복사하는 것이 좋습니다. 일반적으로 OS/400 프로그램 또는 프로시듀어는 OS/400 PASE 어플리케이션에서 직접 실행할 수 없습니다. 자세한 정보는 27 페이지의『OS/400 PASE 프로그램에서 OS/400 프로그램 및 프로시듀어 호출』을 참조하십시오.

OS/400 제공 헤더 파일은 다음 디렉토리에 있습니다.

```
/QIBM/include
```

어플리케이션에 OS/400 API 헤더 파일이 필요한 경우 변환된 파일을 AIX 디렉토리에 복사하기 전에 먼저 이 헤더 파일을 EBCDIC에서 ASCII로 변환해야 합니다.

EBCDIC 텍스트 파일을 ASCII로 변환하는 한 가지 방법은 OS/400 PASE `Rfile` 유틸리티를 사용하는 것입니다.

다음 예에서는 OS/400 PASE `Rfile` 유틸리티를 사용하여 OS/400 헤더 파일 `/QIBM/include/qusec.h`를 읽고 자료를 OS/400 PASE CCSID로 변환하여 각 행에서 후행 공백을 제거한 후, 바이트 스트림 파일 `ascii_qusec.h`에 결과를 기록합니다.

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

내보내기 파일 복사

iSeries 서버에서 AIX 디렉토리로 내보내기 파일을 복사하십시오.

다음 OS/400 디렉토리에 있는 내보내기 파일은 OS/400 고유 함수에 액세스해야 할 어플리케이션을 빌드하는 좋은 방법입니다.

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

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

OS/400 PASE는 다음과 같은 내보내기 파일을 제공합니다.

as400_libc.exp	libc.a에 있는 OS/400 고유 함수에 대한 내보내기 파일 as400_libc.exp 파일은 해당 라이브러리의 AIX 버전에서 내보내지 않은 libc.a의 OS/400 PASE 버전에서 모든 내보내기를 정의합니다.
libdb400.exp	OS/400 데이터베이스 함수에 대한 내보내기 파일 libdb400.exp 파일은 OS/400 PASE libdb400.a 라이브러리에서 내보내기를 정의합니다(iSeries용 DB2 UDB CLI(Call Level Interface) 지원).

파일 복사에 대한 자세한 정보는 iSeries 서버로 OS/400 PASE 프로그램 복사를 참조하십시오.

OS/400 기능에 액세스하는 OS/400 PASE API

OS/400 PASE는 ILE 코드 및 기타 OS/400 기능에 액세스할 수 있는 다양한 API를 제공합니다. 컴파일러가 수행하는 것과 비교하여 사용자 스스로의 준비와 구조 빌드를 수행하려는 정도에 따라 사용할 API를 선택할 수 있습니다. 자세한 정보는 OS/400 PASE API를 참조하십시오.

OS/400 환경에서 OS/400 PASE 프로그램 사용

OS/400 PASE 프로그램은 작업에서 실행 중인 다른 OS/400 프로그램을 호출할 수 있고, 다른 OS/400 프로그램은 OS/400 PASE 프로그램에서 프로시듀어를 호출할 수 있습니다. OS/400 PASE 프로그램을 컴퓨팅 환경에 통합하는 방법에 대한 정보는 다음 주제를 참조하십시오.

OS/400 PASE 프로그램 및 프로시듀어 실행
작업에서 OS/400 PASE 프로그램 시작 및 ILE 프로그램에서 OS/400 PASE 프로시듀어를 호출하는 정보와 예를 제공합니다.

OS/400 PASE 프로그램에서 OS/400 프로그램과 프로시듀어 호출
OS/400 PASE 프로그램에서 ILE 프로시듀어, OS/400 프로그램 및 CL 명령 호출에 대한 정보와 예를 제공합니다.

OS/400 PASE 프로그램과 OS/400의 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 PASE 프로그램은 여러 가지 방식으로 실행할 수 있습니다.

- OS/400 작업에서
- OS/400 PASE 대화식 쉘 환경에서
- ILE 프로시듀어에서 호출된 프로그램으로서

OS/400 PASE 프로그램을 OS/400에서 실행할 경우 OS/400 PASE 환경 변수는 ILE 환경 변수와 독립적입니다. 한 환경에 변수를 설정해도 다른 환경에는 적용되지 않습니다. 환경 변수에 대한 작업은 OS/400 PASE 환경과 OS/400 환경과의 대화 방법에 대해 설명합니다.

OS/400 PASE 프로그램에 대해 작업하게 하는 ILE 프로시듀어

OS/400 PASE는 ILE 코드를 OS/400 PASE 서비스에 액세스할 수 있게 하는(OS/400 PASE 프로그램의 특수 프로그래밍 없이) 여러 ILE 프로시듀어 API를 제공합니다.

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

자세한 정보는 OS/400 PASE ILE 프로시듀어 API를 참조하십시오.

ILE 스레드에 접속

OS/400 PASE로 작성하지 않은 스레드(예: Java 스레드 또는 ILE pthread_create로 작성한 스레드)에서 실행되는 ILE 코드를 통해 OS/400 PASE 프로그램에서 프로시듀어를 호출할 수 있습니다. Qp2CallPase는 ILE 스레드를 OS/400 PASE에 자동 접속하지만(해당 OS/400 PASE pthread 구조 작성) OS/400 PASE 프로그램이 시작될 때 OS/400 PASE 환경 변수 PASE_THREAD_ATTACH를 Y로 설정한 경우에만 가능합니다.

OS/400 PASE에서 OS/400 프로그램으로 결과 리턴

OS/400 _RETURN() 함수를 사용하여 OS/400 PASE 프로그램을 호출하고 OS/400 PASE 환경을 종료하지 않고 결과를 리턴할 수 있습니다. 이를 통해 OS/400 PASE 프로그램을 시작하고, QP2SHELL2(QP2SHELL이 아님) 또는 Qp2RunPase API가 리턴한 후 Qp2CallPase를 사용하여 해당 프로그램에서 프로시듀어를 호출할 수 있습니다.

OS/400 PASE 프로그램 실행을 위한 프로그램 및 프로시듀어

OS/400 PASE는 OS/400 PASE 프로그램을 실행하기 위한 다음과 같은 프로그램 및 프로시듀어를 제공합니다.

『QP2SHELL()을 사용하여 OS/400 PASE 프로그램 실행』
이 OS/400 프로그램을 사용하여 호출된 작업에서 OS/400 PASE 프로그램을 실행하십시오.

23 페이지의『QP2TERM()을 사용하여 OS/400 PASE 프로그램 실행』
이 OS/400 프로그램을 사용하여 대화식 쉘 환경에서 OS/400 PASE 프로그램을 실행하십시오.

23 페이지의『OS/400 프로그램에서 OS/400 PASE 프로그램 실행』
시작할 다른 ILE 프로시듀어 내에서 Qp2RunPase() ILE 프로시듀어를 호출하고 OS/400 PASE 프로그램을 실행하십시오.

25 페이지의『OS/400 프로그램에서 OS/400 PASE 프로시듀어 호출』
다른 ILE 프로시듀어의 Qp2CallPase() 및 Qp2CallPase2() ILE 프로시듀어를 호출하여 이미 OS/400 PASE 환경을 실행 중인 작업에서 OS/400 PASE 프로그램을 실행하십시오.

26 페이지의『Java에서 OS/400 PASE 원시 메소드 사용』
Java 프로그램의 OS/400 PASE 환경에서 실행되는 OS/400 PASE 원시 메소드를 사용하십시오.

QP2SHELL()을 사용하여 OS/400 PASE 프로그램 실행

OS/400 PASE 쉘 프로그램(QP2SHELL 또는 QP2SHELL2)을 실행하여 OS/400 명령행과 고급 언어 프로그램, 일괄처리 작업 또는 대화식 작업에서 OS/400 PASE 프로그램을 실행하십시오. 이들 프로그램은 호출된 작업에서 OS/400 PASE 프로그램을 실행합니다. OS/400 PASE 프로그램의 이름은 프로그램의 매개변수로 전달됩니다 이 프로그램의 사용 방법에 대한 자세한 내용은 QP2SHELL() 및 QP2SHELL2() 설명을 참조하십시오.

QP2SHELL() 프로그램은 새 활성 그룹에서 OS/400 PASE 프로그램을 실행합니다. QP2SHELL2() 프로그램은 호출자의 활성 그룹에서 실행됩니다.

다음 예에서는 OS/400 명령행의 ls 명령을 실행합니다.

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

CL 변수를 사용하여 QP2SHELL()에 값 전달

CL 변수를 사용하여 QP2SHELL()에 값을 전달하는 경우 반드시 변수를 널(null)로 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 화면에 기록합니다.

```
call qp2term
```

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

Korn 쉘이 디폴트이지만, 프로그램으로 전달할 인수 스트링 뿐만 아니라 실행할 OS/400 PASE 프로그램의 경로명도 선택적으로 지정할 수 있습니다.

QP2TERM()으로 시작한 대화식 세션에서 OS/400 PASE 프로그램과 유틸리티를 실행할 수 있습니다. stdout와 stderr이 단말기 화면에 기록되고 화면이동됩니다.

OS/400 프로그램에서 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 호출된 작업에서 OS/400 PASE 프로그램을 실행합니다. OS/400 PASE 프로그램(필요한 공유 라이브러리 포함)을 로드한 다음 프로그램으로 제어를 전송합니다.

이 API는 QP2SHELL() 및 QP2TERM()보다 OS/400 PASE가 실행되는 방식에 있어 더 많은 제어를 제공합니다.

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

예: OS/400 프로그램에서 OS/400 PASE 프로그램 실행: 다음 예는 OS/400 PASE 프로그램을 호출하는 ILE 프로그램 및 ILE 프로그램이 호출하는 OS/400 PASE 프로그램을 보여줍니다.

예 1: OS/400 PASE 프로그램을 호출하는 ILE 프로그램

다음 ILE 프로그램(면책사항 참조)은 OS/400 PASE 프로그램을 호출합니다. 이 예 다음에는 이 프로그램이 호출하는 OS/400 PASE 코드에 대한 예가 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 */
 }
```

예 2: ILE 프로그램에서 호출되는 OS/400 PASE 프로그램

다음 OS/400 PASE 프로그램(면책사항 참조)은 위에 있는 ILE 프로그램으로 호출됩니다.

```
#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 프로그램에서 OS/400 PASE 프로시듀어 호출

Qp2RunPase() API가 먼저 시작되어 작업에서 OS/400 PASE 프로그램을 실행합니다. OS/400 PASE가 이미 해당 작업에서 사용 중이면 오류를 리턴합니다.

이미 OS/400 PASE 프로그램을 실행 중인 작업에서 OS/400 PASE 프로시듀어를 호출하도록 Qp2CallPase() 및 Qp2CallPase2() API를 사용합니다.

Qp2CallPase() API 사용 방법에 대한 예는 예제 프로그램을 참조하십시오.

예: OS/400 프로그램에서 OS/400 PASE 프로시듀어 호출: 다음 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;
}
```

Java에서 OS/400 PASE 원시 메소드 사용

Java 프로그램에서 OS/400 PASE 환경에서 실행되는 OS/400 PASE 원시 메소드를 사용할 수 있습니다. OS/400 PASE 원시 메소드에 대한 지원은 OS/400 PASE 원시 메소드에서 원시 iSeries JNI(Java Native Interface) 전체를 사용하고, 원시 iSeries JVM에서 OS/400 PASE 원시 메소드를 호출하는 기능을 포함합니다.

자세한 정보와 예는 Java에 대한 IBM OS/400 PASE 원시 메소드를 참조하십시오.

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 프로그램의 실행에 사용하는 메소드에 따라 ILE에서 OS/400 PASE로 변수를 복사할 수 있습니다.

대화식 OS/400 PASE 세션의 환경 변수

ILE 환경 변수는 QP2SHELL() 및 QP2TERM()으로 시작될 경우에만 OS/400 PASE에 전달됩니다. WRKENVVAR(환경 변수에 대한 작업) 명령을 사용하여 OS/400 PASE를 시작하기 전에 필요에 따라 환경 변수를 변경, 추가 또는 삭제하십시오.

호출된 OS/400 PASE 세션의 환경 변수

Qp2RunPase() API를 사용하여 OS/400 PASE가 프로그램 호출에서 시작되면 환경 변수에 대한 완전한 제어를 할 수 있습니다. 그러면 OS/400 PASE 프로그램이 호출된 ILE 환경과 관계없는 환경 변수를 전달할 수 있습니다.

CL 명령을 실행하기 전에 ILE에 환경 변수 복사

systemCL 런타임 함수의 옵션을 사용하여 CL 명령을 실행하기 전에 OS/400 PASE 환경 변수를 ILE 환경에 복사할 수 있습니다. 이는 또한 OS/400 PASE `system` 유틸리티의 디폴트 값이기도 합니다.

자세한 정보는 OS/400 PASE 환경 변수 주제를 참조하십시오.

OS/400 PASE 프로그램에서 OS/400 프로그램 및 프로시듀어 호출

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

OS/400 프로그램 및 프로시듀어에 대한 일반 구성 요구사항

OS/400 PASE 프로그램 환경에서 OS/400 환경으로 호출할 경우 다음과 같은 이유 때문에 보통 OS/400 프로그램이 활성 그룹에서 *CALLER로 컴파일되었는지를 확인해야 합니다.

- OS/400 PASE(Qp2RunPase API에서 호출)를 시작한 활성 그룹에서 실행되는 코드만이 Qp2CallPase와 같은 ILE API를 사용하여 OS/400 PASE 프로그램과 대화할 수 있습니다.
- 멀티스레드 작업에서 활성 그룹을 제거해야 할 경우(OS/400 PASE `fork`로 작성된 모든 작업에서 멀티스레드가 가능한 경우) ILE 런타임이 전체 작업을 종료할 수 있습니다(OS/400 PASE도 종료됨). ACTGRP(*CALLER)를 사용하면 종료하려는 시점까지 작업 종료를 지연할 수 있습니다.

멀티스레드가 가능하지 않은 별도의 작업에서 CL 명령(CALL 명령 포함)을 실행하는 `systemCL` 런타임 함수를 사용함으로써 멀티스레드가 가능한 작업에서 실행되는 문제를 방지할 수 있습니다.

OS/400 PASE 환경에서 호출

다음 주제에서는 OS/400 PASE 환경에서의 호출하는 방법에 대한 지침 및 예를 제공합니다.

ILE 프로시듀어 호출
OS/400 PASE에서 ILE 프로시듀어를 호출하기 전에 ILE 프로시듀어가 OS/400 PASE 프로그램의 호출을 처리하도록 설정되었는지를 확인해야 합니다. 또한 컴파일된 AIX 프로그램에서 프로그램 변수 및 구조를 설정해야 합니다.

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

OS/400 명령 실행
OS/400 PASE 프로그램의 CL 명령을 이 프로그램에서 실행할 수 있습니다.

ILE 프로시듀어 호출

OS/400 PASE 프로그램에서 ILE 프로시듀어를 호출하는 경우 텍스트를 적절한 CCSID로 변환하고 변수와 구조를 설정하여 먼저 teraspace에서 프로시듀어를 사용할 수 있도록 준비해야 합니다.

teraspace에서 ILE 프로시듀어 사용 가능

*YES로 설정된 teraspace 옵션을 사용하여 OS/400 PASE에서 호출하는 모든 ILE 모듈을 컴파일해야 합니다. 이 방식으로 ILE 모듈을 컴파일하지 않으면 OS/400 PASE 어플리케이션의 작업 기록부에 MCH4433 오류 메세지(목표 프로그램 &2의 기억장치 모델이 유효하지 않음)가 수신됩니다. 자세한 정보는 ILE 개념 을 참조하십시오.

적절한 CCSID로 텍스트 변환

ILE와 OS/400 PASE 사이에서 전달될 텍스트는 전달되기 전에 먼저 적절한 CCSID로 변환되어야 합니다. 이러한 변환을 수행하지 않을 경우 문자 변수에 해독 불가능한 값이 포함될 수 있습니다.

변수 및 구조 설정

OS/400 PASE 프로그램에서 ILE를 호출하려면 변수 및 구조를 설정해야 합니다. 필수 헤더 파일이 AIX 시스템에 복사되었는지 확인하고 서명, 결과 유형 및 인수 리스트 변수를 설정해야 합니다.

- **헤더 파일:** OS/400 PASE 프로그램에 ILE 호출에 필요한 헤더 파일 as400_types.h 및 as400_protos.h가 있어야 합니다. as400_type.h 헤더 파일에는 OS/400 고유 인터페이스에 사용되는 유형에 대한 정의가 들어 있습니다.
- **서명:** 서명 구조에 OS/400 PASE와 ILE 사이에서 전달되는 인수의 유형 및 그 전달 순서에 대한 설명이 있습니다. 호출 중인 ILE 프로시듀어가 요구하는 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 pragma가 없으면 ILE에 대한 표준 C 링크에서 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 항목에 지정된 유형 필드가 올바른 순서로 구성된 구조여야 합니다. size_ILEarglist() 및 build_ILEarglist() API를 사용하여 서명에 기초한 인수 리스트를 동적으로 빌드할 수 있습니다.

OS/400 PASE 프로그램에서 ILE 프로시듀어를 호출하려면 코드에서 다음 API를 호출하십시오.

1. OS/400 PASE를 시작한 프로시듀어와 연관된 ILE 활성 그룹으로 바인드 프로그램을 로드하십시오. 이 작업을 수행하려면 _ILELOAD() API를 사용하십시오. 바인드 프로그램이 OS/400 PASE를 시작한 활성 그룹에서 이미 활성화된 경우에는 이 단계를 수행하지 않아도 됩니다 이런 경우 활성화 마크 매개변수에 지정된 0 값을 사용하여 현재 활성 그룹의 모든 활성 바인드 프로그램에서 모든 기호를 탐색하면 _ILESYM 단계로 진행할 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. ILE 프로시듀어를 호출하여 OS/400 PASE 프로그램에서 ILE 프로시듀어로 제어를 전송하십시오 이 작업을 수행하려면 _ILECALL() 또는 _ILECALLX() API를 사용하십시오.

OS/400 PASE에서 ILS 프로시듀어를 호출하는 프로세스를 설명한 예는 예: ILE 프로시듀어 호출을 참조하십시오.

예: ILE 프로시듀어 호출: 다음 코드 예제(면책사항 참조)는 서비스 프로그램의 일부인 ILE 프로시듀어 및 프로그램을 작성하는 컴파일러 명령을 호출하는 OS/400 PASE 코드를 표시합니다. 예제에는 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 메소드를 사용하여 _ILECALL API에 자료 구조를 빌드하는 방법을 설명합니다. 그런 후 두 번째 프로시듀어는 인수 리스트를 수동으로 빌드합니다.

예 1: OS/400 PASE C 코드

다음의 코드 예제에는 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 정적 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. 이 예에서는
  * 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 정적 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 예에서는
  * 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 정적 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);
        }
    }
}
```

예 2: ILE C 코드

이 예의 ILE C 코드를 OS/400 시스템에 기록할 수 있습니다. 코드를 작성할 라이브러리에서 소스 실제 파일이 필요합니다. ILE 예에는 주석이 곳곳에 있습니다. 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;
}
```

예 3: 프로그램을 작성하는 컴파일러 명령

OS/400 PASE 프로그램을 컴파일하는 경우 컴파일러 옵션 -qalign=natural 및 -qldbl128을 사용하여 상대 16바이트 정렬을 long double 유형으로 만들어야 합니다. 이 유형은 ILEpointer 유형 내부에서 사용됩니다. 이 정렬은 OS/400의 ILE에 필요합니다. -bI: 옵션의 경우에는 as400_libc.exp 파일을 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 어플리케이션을 작성할 경우 기존 OS/400 프로그램(*PGM 오브젝트)을 활용할 수 있습니다. 또한 systemCL 함수를 사용하여 CL CALL 명령을 실행할 수 있습니다. 관련 정보 및 예는 OS/400 PASE의 OS/400 명령 실행을 참조하십시오.

OS/400 PASE 프로그램에서 OS/400 프로그램을 호출하려면 _PGMCALL 런타임 함수를 사용하십시오. 이 메소드는 systemCL 런타임 함수보다 빠른 처리를 제공하지만 PGMCALL_ASCII_STRINGS를 지정하지 않은 경우 문자 스트링 인수의 자동 변환을 수행하지 않고 다른 작업에서 프로그램을 호출하는 기능을 제공하지 않습니다.

_PGMCALL 런타임 함수를 사용하여 OS/400 PASE 프로그램에서 명령을 호출하는 방법에 대한 예는 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.

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

```
/* 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 기능을 확장할 수 있습니다.

OS/400 PASE 프로그램에서 OS/400 명령을 실행하려면 systemCL 런타임 함수를 사용하십시오.

OS/400 PASE에서 OS/400 명령을 실행하면 systemCL 런타임 함수가 문자 스트링 인수의 ASCII 대 EBCDIC 변환을 처리하므로 다른 작업에서도 이 프로그램을 호출할 수 있습니다.

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 예(면책사항 참조)는 OS/400 PASE 프로그램의 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 방법

OS/400 PASE 프로그램이 OS/400 기능을 사용하도록 사용자 정의할 때 사용자 프로그램이 이들 기능과 대화하는 방식을 고려해야 합니다. 다음 주제는 기본 지침 및 iSeries Information Center에 있는 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 AIX와 동일한 구문을 지원합니다. 그러나 모든 세부사항이 다른 UNIX 시스템과 일치하지는 않습니다.

OS/400 PASE 소켓 지원은 소켓의 AIX 구현과 유사하지만 OS/400 PASE는 OS/400 소켓 구현(AIX 커널의 소켓 구현 대신)을 사용하므로 AIX 작동 방식에서 약간의 차이가 발생합니다.

OS/400 소켓 구현은 UNIX 98 및 BSD(Berkeley Software Distribution) 소켓을 모두 지원합니다. 대부분의 경우 OS/400 PASE는 AIX 구현의 작동을 채택하여 이러한 스타일의 차이점을 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 정보는 소켓 프로그래밍 주제를 참조하십시오. 특히 BSD(Berkeley Software Distribution) 호환성 및 UNIX 98 호환성 주제를 참조하십시오.

데이터베이스

OS/400 PASE는 iSeries용 DB2 UDB CLI(Call Level Interface)를 지원합니다. AIX 및 OS/400에서 DB2 CLI는 서로에 대해 적절한 서브세트가 아니므로 일부 인터페이스에 약간의 차이점이 있고, 구현된 일부 API가 다른 시스템에는 없을 수 있습니다. 이 때문에 다음 사항을 고려해야 합니다.

- 코드를 생성할 수는 있지만 AIX 자체에서 테스트할 수 없습니다. 대신 OS/400 PASE의 플랫폼에서 코드를 테스트해야 합니다.
- 헤더 파일 `sqlcli.h`의 OS/400 버전으로 컴파일해야 합니다. 이 헤더 파일의 AIX 버전을 사용하여 컴파일한 프로그램은 OS/400 PASE에서 실행되지 않습니다.

OS/400은 기본적으로 EBCDIC 코드화 시스템이지만 AIX는 ASCII에 기반합니다. 이러한 차이 때문에 OS/400 데이터베이스(iSeries용 DB2 UDB)와 OS/400 PASE 어플리케이션 간 자료 변환이 종종 필요합니다.

OS/400 CLI를 OS/400 PASE에서 구현하는 경우 OS/400 PASE 제공 라이브러리 루틴은 문자 자료에 대해 ASCII에서 EBCDIC로 또는 그 반대로 자료 변환을 자동 수행합니다. 액세스 중인 자료의 태그 CCSID 및 OS/400 PASE 프로그램이 실행 중인 ASCII CCSID를 기반으로 변환이 수행됩니다. 데이터베이스가 CCSID 65535를 사용하여 태그되는 경우 자동 변환이 수행되지 않습니다. 자료의 코드화 형식을 이해하고 필요한 변환을 수행하는 것은 어플리케이션입니다.

CCSID에 대한 작업

Qp2RunPase() API를 사용할 때 OS/400 PASE CCSID를 명시적으로 지정해야 합니다.

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

- PASE_LANG
- QIBM_PASE_CCSID

ILE 환경이 이들 변수 중 하나라도 생략하면 기본적으로 QP2TERM, QP2SHELL 및 QP2SHELL2는 OS/400 PASE CCSID 및 OS/400 PASE 환경 변수 `LANG` 값을 작업의 CCSID 속성 및 언어에 가장 적합한 OS/400 PASE로 설정합니다.

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

libc.a로의 확장 기능를 통해 OS/400 PASE 어플리케이션이 _SETCCSID() 함수를 사용하여 어플리케이션에서 실행 중인 CCSID를 변경할 수 있습니다.

다른 확장을 통해 OS/400 PASE 어플리케이션이 어플리케이션의 CCSID를 변경하지 않고도 DB2 CLI 내부 변환을 대체할 수 있는 기능을 제공합니다. SQLOverrideCCSID400() 함수는 대체 CCSID의 정수를 단일 매개변수로 승인합니다.

주: CCSID 대체 함수 SQLOverrideCCSID400()이 효력을 가지려면 다른 SQLx() API보다 먼저 호출되어야 합니다. 그렇지 않으면 요구가 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 AIX 시스템에 복사해야 합니다. DB2 CLI 라이브러리 루틴은 OS/400 PASE 환경의 경우 `libdb400.a`에 있으며 `pthread` 인터페이스를 사용하는 해당 루틴이 구현되어 스레드세이프를 제공합니다. 대부분의 OS/400 PASE CLI 함수는 이에 상응하는 ILE CLI 함수를 호출하여 원하는 조작을 수행합니다.

주: OS/400 PASE 프로그램에서 DB2 CLI를 사용할 경우 다음을 고려하십시오.

- SQLGetSubString은 CLOB/DBCLOB 필드를 서브 스트링할 때 항상 EBCDIC 스트링을 리턴합니다. SQLGetSubString은 LOB 자료 유형에만 사용됩니다.
- 결과 세트(표 유형)의 네 번째 열 SQLTables은 항상 EBCDIC로 리턴됩니다.
- OS/400 PASE 프로그램에서 그래픽 유형 자료를 표시하려면 자료가 프로그램에서 `wchar`로 입력되어야 합니다. 그렇지 않으면 데이터베이스는 자료의 CCSID와 OS/400 작업의 CCSID 사이에서 변환됩니다 데이터베이스는 Qp2RunPase() API 또는 SQLOverrideCCSID400() API에서 EBCDIC 그래픽과 CCSID 간의 변환을 지원하지 않습니다.

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

iSeries용 DB2 UDB SQL 호출 레벨 인터페이스를 사용하여 OS/400 PASE가 iSeries용 DB2 UDB에 액세스하는 방법에 대한 예는 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 열이 MN과 일치하는 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 CCSID(코드화 문자 세트 ID) 값을 지정하여 오브젝트의 문자 자료에 고유한 코드화를 식별할 수 있습니다.

OS/400 PASE 바이트 스트림 파일에는 OS/400 PASE 외부의 대부분의 시스템 인터페이스가 필요한 경우 파일에 쓰거나 파일에서 읽어오는 텍스트 자료를 변환하는 데 사용하는 CCSID 속성이 있습니다. OS/400 PASE는 스트림 파일(AIX와 일치)에 쓰거나 이 파일에서 읽은 자료에 대한 CCSID 변환을 수행하지 않지만, 다른 시스템 함수가 파일의 ASCII 텍스트를 올바르게 처리할 수 있도록 OS/400 PASE 프로그램에서 작성한 모든 바이트 스트림 파일의 CCSID 속성을 현재 OS/400 PASE CCSID 값으로 설정합니다.

OS/400 PASE 공유 라이브러리에 제공된 AIX API를 사용하면 OS/400 PASE가 대부분의 자료 변환을 처리합니다. OS/400 PASE 프로그램은 OS/400 PASE 런타임으로 자동 처리되지 않은 모든 문자 자료 변환에 대해 공유 라이브러리 libiconv.a에서 제공된 iconv 함수를 사용할 수 있습니다. 예를 들어 OS/400 PASE 어플리케이션은 주로 OS/400 API 함수()를 호출하기 전에 _ILECALLX 또는 _PGMCALL을 사용하여 문자 스트링을 EBCDIC로 변환해야 합니다.

파일 시스템

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

버퍼링된 입력 및 출력

외부 장치 간의 입력 및 출력은 OS/400에서 버퍼링됩니다 이와 반대로 UNIX 시스템은 일반적으로 문자 단위(버퍼링되지 않음) 입출력 방식으로 작동합니다. OS/400에서는 특정 입력 및 출력 신호(예: Enter, 기능 키 및 시스템 요구)만 시스템에 인터럽트를 송신할 수 있습니다.

자료 변환 지원

OS/400 PASE 프로그램은 ASCII(또는 UTF-8) 경로명을 open 함수에 전달하여 바이트 스트림 파일을 엽니다. 여기서 파일 이름은 OS/400에서 사용된 코드화 체계로 자동 변환되지만 열린 파일에서 읽거나 기록된 자료는 변환되지 않습니다.

자료 변환에 대한 자세한 정보는 자료 코드화를 참조하십시오.

파일 설명자 사용

OS/400 PASE 런타임은 대개 stdin, stdout 및 stderr 파일에 대한 ILE C 런타임 지원을 사용하여 OS/400 PASE 및 ILE 프로그램에 일관성 있는 조작을 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 항상 stdin, stdout 및 stderr에 대한 통합 파일 설명자를 사용하지는 않으므로 OS/400 PASE는 OS/400 PASE 파일 설명자와 통합 파일 시스템의 설명자 간의 맵핑을 제공합니다. 이러한 맵핑으로 인해 OS/400 PASE 프로그램 및 ILE C 프로그램은 서로 다른 설명자 번호를 사용하여 동일한 열린 파일에 액세스할 수 있습니다.

fcntl 함수 F_MAP_XPFFD에서 OS/400 PASE 확장 기능을 사용하여 OS/400 PASE 설명자를 ILE 번호에 할당할 수 있습니다. OS/400 PASE 어플리케이션이 OS/400 PASE에서 작성하지 않은 ILE 설명자에 대한 파일 조작을 수행해야 하는 경우에 유용합니다.

fstatx 함수로의 OS/400 고유 확장 기능 STX_XPFFD_PASE를 통해 OS/400 PASE 프로그램은 OS/400 PASE 파일 설명자에 대한 통합 파일 시스템 설명자 번호를 판별할 수 있습니다. stdin, stdout 및 stderr 파일에 대한 ILE C 런타임 지원에 첨부된 임의의 OS/400 PASE 설명자에 특수 값(음수)이 리턴됩니다.

Qp2RunPase() API가 호출될 때 ILE 환경 변수 QIBM_USE_DESCRIPTOR_STDIO가 Y 또는 I로 설정되는 경우 OS/400 PASE가 파일 설명자 0, 1 및 2를 통합 파일 시스템과 동기화하면 OS/400 PASE 및 ILE C 프로그램은 stdin, stdout 및 stderr 파일에 동일한 설명자 번호를 사용할 수 있습니다. 이 모드에서 작동 중일 때 OS/400 PASE 코드 또는 ILE C 코드가 파일 설명자 0, 1 또는 2를 닫거나 다시 여는 경우 이 변경사항은 두 환경에서 stdin, stdout 및 stderr 처리에 영향을 미칩니다.

OS/400 PASE 런타임은 OS/400 PASE 파일 설명자(소켓 포함)를 통해 읽거나 기록된 자료에서 문자 코드화 변환을 수행하지 않습니다. (단 ILE C stdin에서 읽거나 ILE C stdout 및 stderr에 기록된 자료의 경우 OS/400 PASE CCSID 및 작업 디폴트 CCSID 사이에서 ASCII 대 EBCDIC 변환이 수행됩니다.)

두 환경 변수는 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 설정되면(2진 자료를 stdin에서 읽고 stdout 또는 stderr에 기록할 수 있음) OS/400 PASE stdin, stdout 및 stderr에서 ASCII 대 EBCDIC 변환을 사용할 수 없습니다. QIBM_PASE_DESCRIPTOR_STDIO에 대한 디폴트는 텍스트의 경우 T입니다. 이 값은 EBCDIC에서 ASCII로 변환이 이루어지게 합니다.

파일 시스템에 대한 자세한 정보는 통합 파일 시스템 주제를 참조하십시오.

국제화

OS/400 PASE 런타임이 AIX 런타임에 기반하므로 OS/400 PASE 프로그램에서는 AIX에 지원되는 로케일, 문자 스트링 조작, 날짜 및 시간 서비스, 메세지 카탈로그 및 문자 코드화 변환 등에 동일한 여러 프로그래밍 인터페이스 세트를 사용할 수 있습니다.

OS/400 PASE는 1바이트 및 복수 바이트 문자 코드화 모두에 대한 지원을 포함하여 어플리케이션이 사용하는 로케일을 관리하고 로케일 관련 함수(예: ctype 및 strcoll)를 수행할 수 있도록 AIX 런타임의 인터페이스를 지원합니다.

OS/400 PASE에는 산업 표준 코드화(코드 세트 ISO8859-x), 코드 세트 IBM-1250 및 코드 세트 UTF-8을 사용하는 많은 국가 및 언어를 지원하는 AIX 로케일의 서브세트가 포함됩니다. OS/400 PASE는 IBM-1252 로케일, ISO 8859-15 로케일(모두 1바이트 코드화 사용) 및 UTF-8 로케일의 세 가지 서로 다른 방식으로 유로를 지원합니다.

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

새 로케일 작성

OS/400 PASE는 새 로케일을 작성하는 유틸리티를 제공하지 않습니다. 그러나 localedef 유틸리티를 사용하면 AIX 시스템에서 OS/400 PASE에 사용할 로케일을 작성할 수 있습니다.

로케일 변경

OS/400 PASE 어플리케이션에서 로케일을 변경할 경우 새 로케일의 코드화와 일치하도록 _SETCCSID 런타임 함수를 사용하여 OS/400 PASE CCSID도 변경해야 합니다. 그렇게 하면 OS/400 PASE 런타임이 모든 문자 자료 인터페이스 인수를 올바로 해석하고 EBCDIC 시스템 서비스를 호출할 때 변환될 수 있습니다. cstoccsid 런타임 함수를 사용하여 코드 세트 이름에 해당하는 CCSID를 판별할 수 있습니다.

OS/400 PASE 런타임은 OS/400 PASE 프로그램으로 작성한 파일에서 CCSID 태그를 현재 OS/400 PASE CCSID 값(프로그램이 시작될 때 또는 최신 _SETCCSID 값을 사용할 때 제공됨)으로 설정합니다.

일본어, 한국어, 중국어 및 대만어를 지원하는 OS/400 PASE 어플리케이션에서 UTF-8 로케일을 사용해야 합니다. OS/400에는 이러한 언어에 대한 다른 로케일이 포함되지만, 시스템에서 IBM-euc*XX* 코드 세트의 코드화와 일치시킬 수 있는 OS/400 PASE CCSID 설정을 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 로케일이 저장되는 위치

OS/400 PASE의 변환 오브젝트와 로케일은 OS/400 언어 피처 코드 패키지에 들어 있습니다. OS/400 PASE를 설치하면 설치된 OS/400 언어 피처와 연관된 로케일만 작성됩니다.

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

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

메세지 서비스

OS/400 PASE 신호와 ILE 신호가 독립적이므로 다른 유형의 신호를 발생시켜 신호 유형에 대한 핸들러를 직접 호출할 수 없습니다. OS/400 PASE Qp2SignalPase() API를 사용하여 수신한 ILE 신호에 해당하는 OS/400 PASE 신호를 게시할 수 있습니다. QP2SHELL() 프로그램과 OS/400 PASE fork() 함수는 항상 모든 ILE 신호를 해당 OS/400 PASE 신호로 맵핑하도록 핸들러를 설정합니다

시스템은 Qp2RunPase, Qp2CallPase 또는 Qp2CallPase2 API를 실행하는 호출의 프로그램 메세지 대기행렬로 송신된 모든 OS/400 예외 메세지를 해당 OS/400 PASE 신호로 자동 변환합니다. 따라서 OS/400 PASE 어플리케이션은 시스템에서 변환하는 OS/400 PASE 신호를 처리함으로써 모든 OS/400 예외를 처리할 수 있습니다.

OS/400 PASE는 OS/400 메세지 처리에 대한 직접적인 제어를 부여하는 다음과 같은 런타임 함수를 제공합니다.

- QMHSNDM
- QMHSNDM1

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

이러한 함수에 대한 자세한 정보는 런타임 함수를 참조하십시오.

OS/400 메세지 지원

OS/400은 다양한 문맥으로 메세지를 지원합니다.

- **작업 기록부**. 작업 기록부에는 실행되거나 컴파일되는 동안 OS/400 또는 어플리케이션에서 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.
- **활동 작업에 대한 작업**. WRKACTJOB(활동 작업에 대한 작업) 명령은 OS/400의 작업 및 작업 스택을 검토하는 데 유용합니다.

OS/400의 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 인쇄 출력

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

다음 예에서는 스트림 파일 mydoc.ps의 내용을 스풀 프린터 장치 파일 QPRINT에 변환되지 않은 ASCII 자료로 기록한 다음 CL LPR 명령을 사용하여 스풀 파일을 다른 시스템으로 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 Distribution) 스타일 장치를 지원합니다. 프로그래밍 관점에서 볼 때 이러한 장치는 AIX에서 작동하는 방식과 동일한 방식으로 OS/400 PASE에서 작동합니다.

OS/400 PASE는 AT&T 스타일 장치에 최대 1024개의 인스턴스와 최대 592개의 BSD 스타일 장치를 허용합니다. 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 PASE에서, 이러한 장치는 다음과 같은 방식으로 구성됩니다.

- 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 규칙뿐만 아니라 BSD 종속 및 BSD 1차 장치의 주요 번호를 알고 있어야 합니다. 다음 예의 쉘 스크립트는 추가로 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 단말기에 대한 자세한 정보는 AIX 문서  웹 사이트를 참조하십시오.

보안

보안 관점에서 OS/400 PASE 프로그램에는 OS/400의 기타 프로그램과 같은 보안 제한사항이 적용됩니다. OS/400 PASE 프로그램을 OS/400에서 실행하려면 통합 파일 시스템에서 AIX 2진에 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 특히 OS/400 PASE 프로그램을 실행할 때 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 동일하게 권한 검사가 제공됩니다.

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

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/passwd`와 같은 파일이 아닌 개별 *profiles*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()` API로 전달하면 시스템은 이름 스트링을 예상되는 문자로 변환합니다.

getpwuid() 또는 getgrgid()를 호출하여 리턴한 프로파일명을 가져오면 결과를 대문자로 리턴하는 OS/400 PASE 환경 변수를 PASE_USRGRP_LOWERCASE=N으로 설정하지 않은 경우 프로파일명은 소문자가 됩니다.

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/400에는 UID가 0인 보안 담당자 QSECOFR에 대한 사용자 프로파일이 있습니다. 그러나 OS/400은 시스템 관리자가 개별 사용자에게 할당할 수 있는 특정 권한 세트도 제공합니다. 이러한 권한 중 하나인 *ALLOBJ는 파일 액세스에 대한 임의의 액세스 제어를 대체합니다(예: UNIX 시스템의 루트 권한을 일반적으로 사용).

루트 액세스를 사용하는 이식된 어플리케이션에서는 *ALLOBJ 권한이 주어지는 *어플리케이션 사용자*에 대한 특정 사용자 프로파일을 작성하여, 단일 어플리케이션이 필요로 하는 것 이상의 권한을 갖는 QSECOFR의 사용을 방지하는 것이 좋습니다. UNIX 시스템과는 달리, OS/400은 사용자를 위한 그룹 멤버쉽이 필요하지 않습니다. OS/400에서 사용자 프로파일에 대한 GID가 0이면 더 많은 권한을 가진 그룹이 아닌 *할당된 그룹 없음*을 의미합니다.

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

OS/400 PASE는 무결성과 보안을 유지할 별도의 주소 공간을 각 프로세스에 부여합니다. OS/400 PASE 주소 공간에서 자원을 사용할 수 없으면 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 요청합니다. OS/400 PASE 프로그램에 대한 시스템 호출은 OS/400에서 처리합니다. 이 인터페이스는 시스템 내부에 간접적이고 안전한 OS/400 PASE 프로그램 액세스만 부여합니다.

iSeries 서버의 보안에 대한 자세한 정보는 보안 주제를 참조하십시오.

작업 관리

OS/400은 시스템에서 다른 모든 작업을 처리하는 것과 같은 방식으로 OS/400 PASE 프로그램을 처리합니다. OS/400이 작업을 처리하는 방법에 대한 자세한 정보는 작업 관리 주제를 참조하십시오.

OS/400 PASE 프로그램 디버그

OS/400 PASE 런타임 환경은 syslog() 런타임 함수에 라이브러리 지원을 제공하고, 더 복잡한 메시지 라우팅을 위해 syslogd 2진을 제공합니다. 또한 OS/400의 기존 자원(예: 진단 메세지 및 OS/400 시스템 오퍼레이터 메세지 대기행렬 QSYSOPR에 송신 중인 심각한 메세지 및 진단 메세지에 대한 작업 기록부)을 사용할 수 있습니다.

어플리케이션에 따라 OS/400 PASE 어플리케이션을 디버그하는 전략이 달라질 수 있습니다.

- 어플리케이션에 OS/400 통합(예: iSeries용 DB2 UDB 또는 ILE 함수와의 통합)이 필요하지 않으면 먼저 AIX에서 어플리케이션을 디버그해야 합니다.
- 그런 다음 OS/400 PASE dbx 및 OS/400 디버그 기능(예: 작업 기록부)의 조합을 사용하여 OS/400에서 어플리케이션을 디버그합니다.

데이터베이스 또는 ILE 함수를 사용하도록 코딩한 어플리케이션을 AIX에서 전부 테스트할 수는 없지만 AIX에서 어플리케이션의 나머지 부분을 디버그하여 적절한 구조 및 설계를 할 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 디버거를 AIX 소스에 표시할 수 있습니다.

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

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

OS/400 디버깅 툴 사용

| OS/400에서 다음 툴을 사용하여 OS/400 PASE 어플리케이션을 디버그할 수 있습니다.

| • iSeries System Debugger는 OS/400 PASE 어플리케이션 디버깅을 위한 특정 지원을 제공합니다.

| • ILE C 소스 디버거는 코드의 문제점을 판별하기 위한 효과적 툴입니다. 이 툴을 학습하려면 WebSphere® Development Studio ILE C/C++ Programmer's Guide 를 참조하십시오.

성능 최적화

최상의 성능을 얻으려면 어플리케이션 2진을 로컬 스트림 파일 시스템에 저장하도록 하십시오. 2진(기본 프로그램 및 라이브러리)이 로컬 스트림 파일 시스템의 외부에 있는 경우 파일 맵핑을 수행할 수 없으므로 OS/400 PASE 프로그램 시작 속도가 훨씬 느려집니다.

많은 fork() 조작을 수행하는 OS/400 PASE에서 어플리케이션을 실행하는 경우 해당 어플리케이션이 AIX에서처럼 빨리 실행되지 않습니다. 각각의 OS/400 PASE fork() 조작이 성능에 상당한 영향을 미칠 수 있는 새 OS/400 작업을 시작하기 때문입니다.

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

예

다음 예는 OS/400 PASE 정보에서 제공되었습니다. 이 예를 사용하기 전에 코드 면책사항 정보를 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 Use DB2 UDB 함수 사용

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

코드 면책사항 정보

이 문서에는 프로그래밍 예제가 들어 있습니다.

IBM은 사용자의 특정 요구에 맞게 유사한 기능을 생성할 수 있도록 모든 프로그래밍 코드 예제를 사용할 수 있는 비독점적인 저작권 라이센스를 부여합니다.

모든 샘플 예제는 IBM에 의해 예시 목적으로만 제공됩니다. 이러한 샘플 프로그램은 모든 조건하에서 완전히 테스트된 것은 아닙니다. 따라서 IBM은 이들 샘플 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증하거나 암시하지 않습니다

여기에 포함된 모든 프로그램은 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여 일체의 보증 없이 "현상태대로" 제공됩니다.

OS/400 PASE 관련 정보

다음은 OS/400 PASE 주제와 관련된 Information Center 주제 및 웹 사이트입니다.

기타 iSeries Information Center 주제

- OS/400 PASE API

  다음 같은 OS/400 PASE API의 일반 범주에 대한 자세한 내용은 이 주제를 참조하십시오.

  – 호출 가능 프로그램 API
  – ILE 프로시듀어 API
  – OS/400 PASE 프로그램에서 사용할 런타임 함수

OS/400 PASE 프로그램을 실행하려면 시스템 API를 호출해야 합니다. 이 시스템은 OS/400 PASE 프로그램을 실행할 호출 가능한 프로그램 API 및 ILE 프로시듀어 API를 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) 및 OS/400 PASE 프로그램으로 실행되는 200개의 유틸리티가 있습니다. OS/400 PASE 쉘 및 유틸리티는 많은 산업 표준 및 사실상의 표준 명령을 포함하는 확장 가능한 스크립팅 환경을 제공합니다.

- OS/400 PASE 명령

  이 주제에서 설명한 대부분의 OS/400 PASE 명령은 AIX 명령과 동일한 옵션 및 작동을 제공합니다. OS/400 PASE 명령과 함께 각 OS/400 PASE 쉘은 여러 내장 명령(예: `cd`, `exec` 및 `if`)을 지원합니다.

- OS/400 PASE 라이브러리

  OS/400 PASE 런타임은 AIX 런타임에서 제공되는 많은 인터페이스 서브세트를 지원합니다. OS/400 PASE에서 지원되는 대부분의 런타임 인터페이스는 AIX와 동일한 옵션 및 작동을 제공합니다. OS/400 PASE 런타임 라이브러리는 `/usr/lib`에 기호 링크로 설치됩니다.

웹 사이트

- Enablement roadmaps & resources (http://www.ibm.com/servers/enable/site/porting/index.html) 이 웹 사이트에서는 해당 어플리케이션을 iSeries 서버로 이식하는 데 있어서 OS/400 PASE를 다른 솔루션과 비교합니다.

- OS/400 PASE (http://www.ibm.com/servers/enable/site/porting/iseries/pase/index.html)이 웹 사이트는 OS/400 PASE에 iSeries 서버로 이식할 어플리케이션에 대한 정보를 제공합니다.

- API 분석 툴 (http://www.ibm.com/servers/enable/site/porting/iseries/overview/apitool.html)이 분석 툴은 OS/400 PASE가 어플리케이션의 AIX 명령, API 및 유틸리티 사용을 지원하는 방법에 대한 자세한 정보를 제공합니다.

- AIX 문서 (http://www.ibm.com/servers/aix/library/) 이 웹 사이트에서는 AIX 명령 및 유틸리티에 대한 정보를 제공합니다.

뉴스 그룹

OS/400 PASE 뉴스 그룹(news://news.software.ibm.com/ibm.software.iseries.pase)은 OS/400 PASE와 관련된 사용자 질문 및 응답을 설명합니다.

IBM 레드북™ 및 Redpaper

Bringing PHP to your iSeries server  이 Redpaper에서 설명한 단계별 구현은 OS/400 PASE(OS/400 Portable Application Solutions Environment)에서 실행되는 하이퍼텍스트 프리프로세서(PHP)의 CGI을 포함합니다.

PDF 파일 저장

PDF를 보거나 인쇄를 위해 워크스테이션에 저장하려면 다음을 수행하십시오.

- 브라우저의 PDF 링크를 마우스 오른쪽 버튼으로 클릭하십시오.
- Internet Explorer를 사용하는 경우 **다른 이름으로 대상 저장...**을 클릭하십시오. Netscape Communicator를 사용하는 경우 **다른 이름으로 링크 저장...**을 클릭하십시오.
- PDF를 저장하려는 디렉토리로 가십시오.
- **저장**을 클릭하십시오.

Adobe Acrobat Reader 다운로드

해당 PDF를 보거나 인쇄하려면 Adobe Acrobat Reader가 필요합니다. Adobe 웹 사이트 (www.adobe.com/products/acrobat/readstep.html)  에서 사본을 다운로드할 수 있습니다.

부록. 주의사항

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다.

IBM은 다른 국가에서는 이 자료에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

| 135-270
| 서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
| 한국 아이.비.엠 주식회사
| 고객만족센터
| 전화번호: 080-023-8080

2바이트(DBCS) 정보에 관한 라이센스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

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

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다. IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 묵시적이든 명시적이든 일체의 보증없이 이 책을 『현상태대로』 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 이 변경사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 언급되는 비IBM 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

| IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용
| 하거나 배포할 수 있습니다.

(1) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (2) 교환된 정보의 상호 이용을 목적으로 정보를 원하는 프로그램 라이센스 사용자는 다음 주소로 문의하십시오.

| 135-270
| 서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
| 한국 아이.비.엠 주식회사
| 고객만족센터

이러한 정보는 해당 조항 및 조건에 따라(예를 들면, 사용료 지불 포함) 사용할 수 있습니다.

이 정보에 기술된 라이센스가 있는 프로그램 및 이 프로그램에 대해 사용 가능한 모든 라이센스가 있는 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

라이센스:

이 정보에는 여러 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원시 언어로 된 샘플 응용프로그램이 들어 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 추가 비용없이 이들 샘플 프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다. 예제는 모든 조건하에서 철저히 테스트된 것은 아닙니다. 따라서 IBM은 이들 프로그램의 신뢰성, 실용성 또는 기능에 대해 보증할 수 없습니다. 귀하는 IBM의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 추가 비용없이 이러한 샘플 응용프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다.

상표

다음 용어는 미국 또는 기타 국가에서 사용되는 IBM Corporation의 상표입니다.

AFS
AIX
AS/400e
DB2
DB2 Universal Database
DFS
IBM
ILE(Integrated Language Environment)
iSeries
OS/400
PartnerWorld
PowerPC

pSeries
Redbooks
VisualAge
WebSphere

Microsoft, Windows, Windows NT 및 Windows 로고는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 상표입니다.

Java 및 모든 Java 기반 상표는 미국 또는 기타 국가에서 사용되는 Sun Microsystems, Inc의 상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 The Open Group의 등록상표입니다.

기타 회사, 제품 또는 서비스 이름은 해당 회사의 상표 또는 서비스표입니다.

서적의 다운로드 및 인쇄 조건

귀하가 다운로드하려는 서적을 사용하는 데에는 다음의 조건이 적용되며 귀하가 이를 승인하는 경우에 해당 서적을 사용할 수 있습니다.

개인적인 사용: 일체의 소유권 표시를 하는 경우에 한하여 귀하는 이들 서적을 개인적이며 비상업적인 용도로 복제할 수 있습니다. 귀하는 IBM의 명시적인 동의없이 해당 서적에 대한 2차적 저작물 또는 그 일부를 배포, 전시 또는 작성할 수 없습니다.

상업적 사용: 일체의 소유권 표시를 하는 경우에 한하여 귀하는 이들 서적을 귀하 사업장 내에서만 복제, 배포 귀하는 IBM의 명시적인 동의없이 귀하의 사업장 이외에서 해당 서적의 2차적 저작물을 작성할 수 없으며 이들 서적 또는 그 일부를 복제, 배포 또는 전시할 수 없습니다.

본 계약에서 명시하지 않는 한, 본 서적 또는 본 서적에 포함된 정보, 데이터, 소프트웨어 또는 기타 지적 재산권에 대하여 다른 허가나 라이센스 또는 권리가 부여되지 않습니다.

해당 서적의 사용이 IBM에게 손해를 가져오거나, 상기 지시사항이 적절하게 준수되지 않은 것으로 IBM이 판단한 경우, IBM은 본 계약에서 부여한 서적에 대해 허가를 취소할 권리가 있습니다.

귀하는 미국 수출법 및 관련 규정을 포함하여 모든 적용 가능한 법률 및 규정을 철저히 준수하지 않는 경우 본 정보를 다운로드, 송신 또는 재송신할 수 없습니다. IBM은 이들 서적의 내용과 관련하여 어떠한 보증도 하지 않습니다. 본 서적은 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 묵시적이든 명시적이든 일체의 보증없이 ″현상태대로″ 제공됩니다.

귀하는 본 사이트로부터 서적을 다운로드하거나 인쇄함으로써 본 조건에 동의한 것으로 간주됩니다.