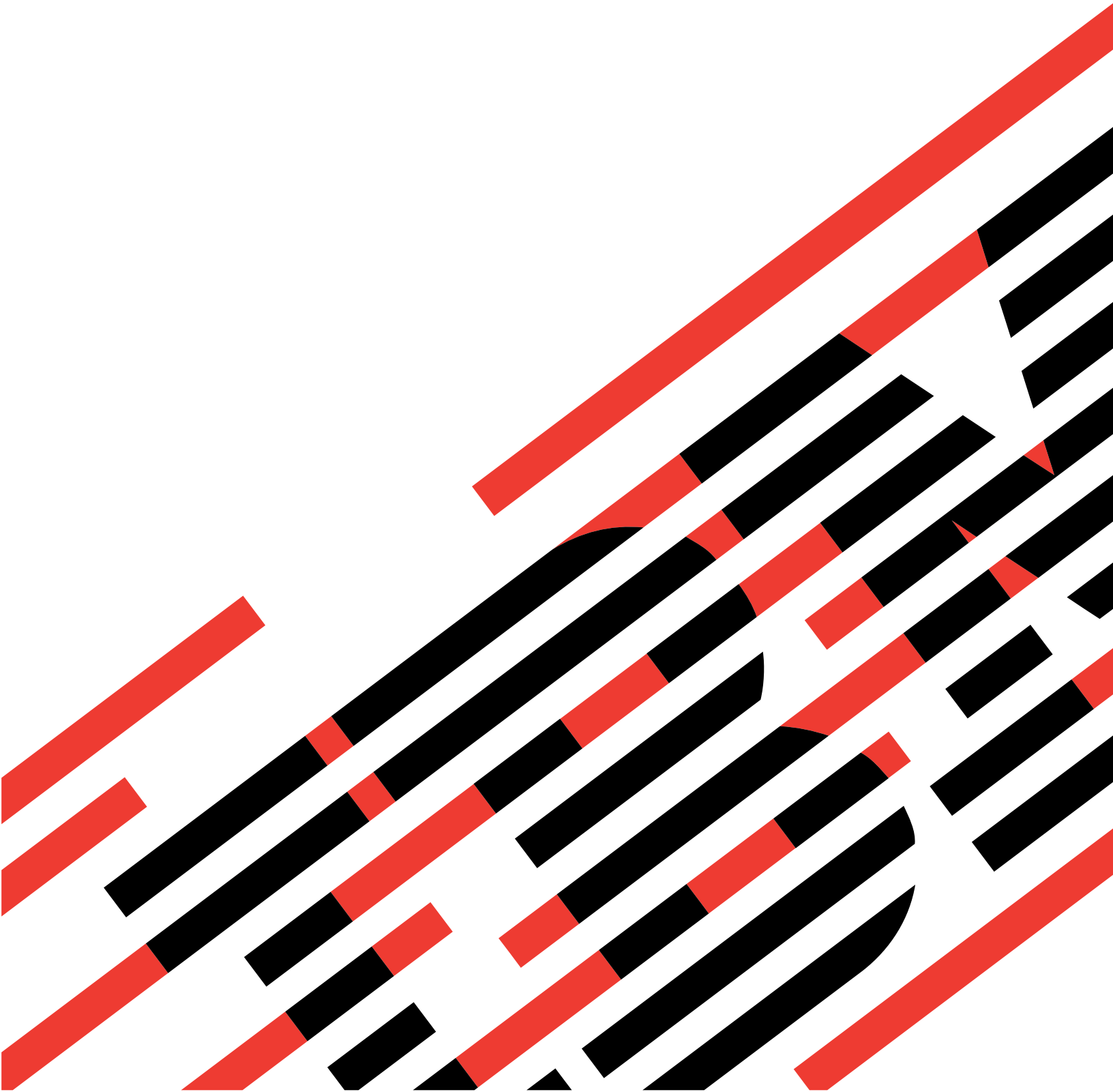
IBM

eserver

iSeries

OS/400 PASE



IBM

@server

iSeries

OS/400 PASE

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

原　典： RZAL–F000–02
iSeries
OS/400 PASE

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2002.11

目次

OS/400 PASE

OS/400(R) ポータブル・アプリケーション・ソリューション環境 (OS/400 PASE) は、最小限の労力で AIX(R) アプリケーションを iSeries(TM) サーバーに移植することを可能にします。 UNIX システムの実行の複雑さに煩わされることなく、選択した UNIX(R) アプリケーションを実行できる統合された実行環境が、OS/400 PASE によって提供されます。 » また、OS/400 PASE には、強力なスクリプト環境を提供する、業界標準のシェルやユーティリティーも備えられています。 «

OS/400 PASE にさらに精通するには、以下のトピックを参考にしてください。このリリースの新機能や、トピックの印刷方法に関する情報も参照できます。

OS/400 PASE の紹介
このトピックでは、OS/400 ポータブル・アプリケーション・ソリューション環境 (PASE) の概要を示し、いつどのように OS/400 PASE が役立つのかを説明し、ご使用の iSeries サーバーに OS/400 PASE をインストールする手順を記載しています。

OS/400 PASE の計画
このトピックでは、OS/400 PASE を実際に使用する前に考慮する必要のある、いくつかの技術的な要件を取り上げています。

OS/400 PASE で実行するプログラムの準備
このトピックでは、OS/400 PASE で効果的に稼働する AIX プログラムの作成、コンパイル、およびコピーの手順を記載しています。

OS/400 環境での OS/400 PASE プログラムの使用
このトピックでは、OS/400 環境で OS/400 PASE を実行する方法、 OS/400 PASE プログラム内から OS/400 プログラムや ILE プロシージャーを呼び出す方法、さらに、セキュリティー、メッセージング、データベース、通信、実行管理、印刷、および統合言語環境 (integrated language environment(R)) (ILE) などの OS/400 機能と OS/400 PASE プログラムがどのように相互作用するかを説明します。

OS/400 PASE のトラブルシューティング
このトピックでは、OS/400 PASE プログラム内のエラーを修正し、プログラムをより効率的かつ効果的に実行させるために使用できる情報を記載しています。

例
このトピックには、この資料に含まれている各サンプルと、そのサンプルを使用する前に読む必要のあるコード・サンプルの特記事項へのリンクがあります。

OS/400 PASE 関連情報
このトピックでは、OS/400 PASE の API、ライブラリー、およびユーティリティーに関して、iSeries Information Center 内の詳細情報の参照先を示しています。さらにこのトピックには、OS/400 PASE や AIX に関係する、 Information Center 外の付加的な情報へのリンクもあります。

V5R2 の新機能

ここでは、いくつかの OS/400 PASE の V5R2 での重要な拡張および変更内容を示します。

- OS/400 PASEは、テストへの要件を解決して、AIX での OS/400 PASE プログラムをコンパイルし、OS/400 PASE 環境で二つの AIX コンパイラー製品のうちのいずれかのインストールをサポートします。 詳細については、AIX ソースのコンパイルを参照してください。
- 疑似端末 (PTY) サポートおよび UNIX-style ジョブ制御。詳細については、疑似端末 (PTY) を参照してください。
- 100 以上の新しいユーティリティー。完全なリストについては、 OS/400 PASE シェルおよびユーティリティーのトピックを参照してください。
- 以下の新しいライブラリーが追加されました。完全なリストについては、 OS/400 PASE ランタイム・ライブラリー (OS/400 PASE Run-time Libraries) のトピックを参照してください。

libcur.a	AIX レガシーの Curses ライブラリー
libg.a	デバッグ・サポート
libgair4.a	内部 X Window サポート
libl.a	lex サポート
libld.a	オブジェクト・ファイルのアクセス・ルーチン・ライブラリー
libm.a	IEEE 数学ライブラリー
libPW.a	プログラマー・ワークベンチ・ライブラリー
libxcurses.a	Curses ライブラリー
libXi.a	X Windows 入力処理
libXtst.a	X Windows テスト・サポート
liby.a	yacc サポート

- 何らかのサポートされていない呼び出しを、OS/400 PASE プログラムが使用した場合、新しいメッセージ (MCH3204) がジョブ・ログに表示されます。このメッセージ・テキストには、そのシステム呼び出しの名前およびそのエラーの原因となった命令アドレスが含まれています。
- 新規および変更された OS/400 PASE ランタイム関数 (OS/400 PASE run-time functions) を以下に示します。
  - _CVTERRNO (OS/400 PASE errno から ILE errno へ変換します)
  - _ILECALLX (拡張された ILE プロシージャー呼び出し)
  - _PMGCALL (OS/400 プログラムを呼び出します)
  - _RETURN (既存の OS/400 PASE なしで戻します)
  - _RSLOBJ、_RSLOBJ2 (OS/400 オブジェクトへ解決します)
  - _STRLEN_SPP、_STRCPY_SPP (16 バイト ILE ポインターを使用したストリング処理)
  - Qp2paseCCSID (OS/400 PASE CCSID を検索します)
  - Qp2jobCCSID (前回の OS/400 PASE CCSID の設定からジョブのデフォルト CCSID を検索します)
  - faccessx
  - fchdir
  - fclear
  - fclear
  - getaddrinfo、getnameinfo
  - getcontext、setcontext
  - getpri、getpriority、setpriority
  - getprocs64、getthrds64
  - gettimer、settimer
  - msem_init、msem_lock、msleep、msem_unlock、 msem_remove

- pread、pwrite
- setgroups
- sigstack、sigaltstack (代替信号スタック)
- statpriv
- statvfs、fstatvfs
- sync
- ustat

• 新規および変更された (ILE) OS/400 PASE 用の API (APIs for OS/400 PASE) を以下に示します。

- QP2SHELL2 (QP2SHELL に似ていますが、呼び出し元の活動化グループで実行されます)
- Qp2ptrsize (OS/400 PASE ポインター・サイズを検索します)
- Qp2paseCCSID (OS/400 PASE CCSID を検索します)
- Qp2jobCCSID (OS/400 PASE CCSID の前回の設定時のジョブのデフォルト CCSID を検索します)
- Qp2errnop (現在のスレッドの OS/400 PASE errno を見つけます)
- Qp2malloc (OS/400 PASE ヒープ・メモリーを割り振ります)
- Qp2free (OS/400 PASE ヒープ・メモリーを解放します)
- Qp2dlopen (OS/400 PASE モジュールを動的にロードします)
- Qp2dlsym (OS/400 PASE dlopen によってオープンされたモジュール内のシンボルを検索します)
- Qp2dlclose (OS/400 PASE dlopen によってロードされたモジュールをクローズおよびアンロードします)
- Qp2dlerror (前回の動的ロード操作のエラー情報を検索します)
- Qp2CallPase (および Qp2CallPase2)。これは、アドレス渡しの引き数および結果を拡張したもので、OS/400 PASE で開始されていないスレッドで OS/400 PASE プロシージャーを呼び出すためのものです。

• OS/400 PASE ロケール (およびグローバリゼーション・サポート用の他のファイル) は、 OS/400 言語フィーチャー・コードとともにパッケージ化されています。詳細については、グローバリゼーションを参照してください。さらに、さまざまなキーボードおよび文字セットの X Windows 処理で使用される、新しい 200 以上のファイルの他に、以下の 65 の新しいロケールがあります。完全なリストについては、 OS/400 PASE ロケール (OS/400 PASE Locales) のトピックを参照してください。

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

```
EN_IN.UTF-8             ar_QA.ISO8859-6         fr_LU.8859-15@euro
EN_NZ.UTF-8             ar_SA.ISO8859-6         sh_YU.ISO8859-2
ES_AR.UTF-8             ar_SY.ISO8859-6         sr_YU.ISO8859-5
ES_CL.UTF-8             ar_TN.ISO8859-6
```

新規および変更箇所の識別方法

技術的な変更が加えられた箇所を識別しやすくするために、以下の情報が使用されています。

- » は、新規および変更された情報の開始箇所を示しています。
- « は、新規および変更された情報の終了箇所を示しています。

新機能あるいは、リリースの変更内容のその他の情報については、 Memo to Users  を参照してください。

トピックの印刷

このトピックの PDF 版をダウンロードし、表示するには、『OS/400 PASE』(約 640 KB、62 ページ) を選択します。

PDF ファイルの保存

表示用または印刷用の PDF ファイルをワークステーションに保存するには、次のようにします。

1. ブラウザーで PDF を開く (上記のリンクをクリックする)。
2. ブラウザーのメニューから**「ファイル」**をクリックする。
3. **「名前を付けて保存」**をクリックする。
4. PDF を保存したいディレクトリーに進む。
5. **「保存」**をクリックする。

Adobe Acrobat Reader のダウンロード

PDF ファイルを表示または印刷するために Adobe Acrobat Reader が必要です。これは、 Adobe の Web サイト (www.adobe.com/products/acrobat/readstep.html)  からダウンロードできます。

OS/400 PASE の紹介

クロス・プラットフォーム・アプリケーションの開発と展開は、有効なビジネス・コンピューティング環境において極めて重要な構成要素です。加えて、システムによって提供される機能が使いやすいことや統合しやすいことも重要です。 iSeries および AS/400e(TM) サーバーはまさに、これらすべての条件を満たしています。ビジネスは日ごとにオープン・コンピューティング環境へと移行しつつあるので、これら多様な目標を達成するには時間や費用がかかり、困難な場合があります。たとえば、AIX オペレーティング・システム上で実行し、その機能を利用する使い慣れたアプリケーションを使用したい、しかし AIX と OS/400 オペレーティング・システムの両方を管理するのは面倒だと思うことがあるかもしれません。

そのようなとき、OS/400 ポータブル・アプリケーション・ソリューション環境 (OS/400 PASE) が役に立ちます。 OS/400 PASE を使用すれば、わずかな変更で、または全く変更しなくても、 OS/400 上の多数の AIX アプリケーション・バイナリーを実行することができ、さらにプラットフォーム・ソリューション・ポートフォリオを効果的に拡張することができます。

OS/400 PASE の詳細については、以下のトピックを参照してください。

- OS/400 PASE の概要
- OS/400 PASE の使用がアプリケーション開発において役立つケース
- OS/400 PASE のインストール

OS/400 PASE の概要

OS/400 ポータブル・アプリケーション・ソリューション環境 (OS/400 PASE) は、 OS/400 上で実行している AIX アプリケーション用の統合ランタイム環境です。これは AIX のアプリケーション・バイナリー・インターフェース (ABI) をサポートしており、 AIX 共用ライブラリー、シェル、およびユーティリティーによって提供されているサポートの幅広いサブセットを提供しています。 OS/400 PASE は PowerPC(TM) のマシン・インストラクションの直接実行をサポートしているため、マシン・インストラクションのエミュレートのみを行う環境の弊害がありません。

OS/400 PASE アプリケーションは、以下の特徴があります。

- C、C++、Fortran、または PowerPC アセンブラーで作成できます。
- AIX PowerPC アプリケーションと同じバイナリー実行可能フォーマットを使用します。
- OS/400 ジョブで実行します。
- ファイル・システム、セキュリティー、およびソケットといった OS/400 システム機能を使用します。

OS/400 PASE は OS/400 での UNIX オペレーティング・システムではないので注意してください。 OS/400 PASE は、ほとんどあるいは全く変更しなくても、 OS/400 上で AIX プログラムを実行するように設計されています。他の UNIX ベース環境からのプログラムは、 OS/400 PASE で実行するための最初のステップとして、 AIX でコンパイルできるように作成する必要があります。

OS/400 PASE 統合ランタイムは、 iSeries サーバー上のライセンス内部コード (LIC) カーネルで実行します。システムは、 OS/400 PASE と他のランタイム環境 (ILE や Java(TM) など) の間で多くの共通 OS/400 機能を統合します。 OS/400 PASE は AIX システム呼び出しの幅広いサブセットをインプリメントします。 » OS/400 PASE のシステム・サポートでは、 OS/400 PASE プログラムがアクセスできるメモリーを制御し、特権のないマシン・インストラクションだけを使用するように制限することにより、システム・セキュリティーを強化します。 «

最小限の労力での迅速なアプリケーション開発

多くの場合、AIX プログラムはほとんどあるいは全く変更せずに、 OS/400 PASE で実行できます。どの程度の AIX プログラミング・スキルが必要かは、 AIX プログラムの設計に応じて異なります。さらに、プログラム設計で (CL コマンドなどで) OS/400 アプリケーションの統合をさらに行うことにより、アプリケーション・ユーザーが構成に向ける注意を最小限にとどめることができます。

OS/400 PASE では、 OS/400 のマーケットでの成功から益を得たいソリューション開発者のために、別の移植オプションを用意しています。 OS/400 PASE が移植時間を著しく短縮する手段を提供することにより、マーケットまでの時間が改善され、ソリューション開発者は投資に見合ったものを得ることができます。

OS/400 での AIX テクノロジーの幅広いサブセット

OS/400 PASE では、以下のような AIX テクノロジーの幅広いサブセットに基づいた、アプリケーション・ランタイムがインプリメントされています。

- 標準の C および C++ ランタイム (スレッド・セーフと非スレッド・セーフの両方)
- Fortran ランタイム (スレッド・セーフと非スレッド・セーフの両方)

- pthreads スレッド化パッケージ
- データ変換用の iconv サービス
- バークレー・ソフトウェア・ディストリビューション (BSD) 同等サポート
- Motif ウィジェット・セットが含まれている X Window システム・クライアント・サポート
- 疑似端末 (PTY) サポート

アプリケーションは、 OS/400 PASE によってサポートされているレベルと互換性のある AIX のレベルで実行している AIX ワークステーションで、開発およびコンパイルされます。それから、OS/400 で実行されます。

OS/400 PASE 内で完全にアプリケーションを開発、コンパイル、ビルド、および実行するために、代替として OS/400 PASE 環境に次の製品のいずれかをインストールできます。

- IBM(R) VisualAge C++ Professional for AIX (バージョン 6)
- IBM C for AIX (バージョン 6)

詳細については、AIX ソースのコンパイル を参照してください。

OS/400 PASE には、 Korn、Bourne、C の各シェルと、強力なスクリプト環境を提供する約 200 個のユーティリティーも含まれています。詳細については、OS/400 PASE シェルおよびユーティリティーを参照してください。

OS/400 PASE は、AIX および OS/400 オペレーティング・システム用の共通プロセッサー・テクノロジーでの IBM インベストメントを使用します。PowerPC プロセッサーは、 OS/400 モードから AIX モードへ切り替えて、 OS/400 PASE ランタイムでアプリケーションを実行します。

OS/400 PASE で実行するアプリケーションは、 OS/400 統合ファイル・システムおよび DB2(R) Universal Database(R) for iSeries に統合されます。それらは Java および統合言語環境 (ILE) アプリケーションを呼び出す (または呼び出される) ことができます。一般にそれらは、セキュリティー、メッセージング、通信、およびバックアップとリカバリーなど、 OS/400 操作環境のすべての面を活用できます。同時に、AIX インターフェースから派生したアプリケーション・インターフェースも活用します。

アプリケーション開発で OS/400 PASE の使用が役立つケース

OS/400 PASE では、 AIX アプリケーションを iSeries サーバーへ移植する方法を、非常に柔軟に決定できます。もちろん、このようなことを行えるのは OS/400 PASE だけです。

API の分析

アプリケーションが OS/400 PASE に適しているかどうかを判別する場合、まず最初に行うのはアプリケーションの分析であり、それが使用する API、ライブラリー、およびユーティリティーと、アプリケーションが OS/400 でどれほど効率的に実行されるかを分析します。 IBM PartnerWorld(TM) チームは、アプリケーションを分析し障害の可能性を述べるフリーな移植評価ツール、 API Analysis Tool を使用してこのエリアの手助けをします。OS/400 PASE にアプリケーションを移植する際のプロシージャーで、この分析ツールをどのように組み込むかについては、 OS/400 PASE で実行するプログラムの準備を参照してください。

OS/400 PASE アプリケーションにすることが可能なものの特性

以下に、OS/400 PASE を使用するかどうかを決定する際に考慮できる、いくつかの有用なガイドラインを示します。

- **AIX アプリケーションで数値計算に重点が置かれているか?**
  OS/400 PASE は、高度に最適化された数学ライブラリーを提供することによって、数値計算のアプリケーションを実行するのに適した環境を備えています。
- **アプリケーションが OS/400 PASE でのみサポートされている (または ILE で一部しかサポートされていない) 機能 (たとえば、 fork()、X Window システム、疑似端末 (PTY) など) にかなり依存しているか?**
  OS/400 PASE は `fork()` および `exec()` のサポートを提供しています。これらは、現在の OS/400 システムにはありません。(ただし、 `spawn()` を使用した場合は例外です。この場合、`fork()` 関数が `exec()` 関数に組み込まれます。)
- **アプリケーションが複雑な AIX ベースで作成されたプロセスおよびテスト環境を使用しているか?**
  OS/400 PASE では、AIX ベースで作成されたプロセスを使用します。これは特に、新しいプラットフォームに移すことが簡単ではない、既存の複雑なプロセスがある場合に役立ちます。
- **アプリケーションが ASCII 文字セットに依存しているか?**
  OS/400 PASE では、これらを必要とするアプリケーションを十分にサポートしています。
- **アプリケーションで多くのポインター操作が行われるか、または整数からポインターへの変換 (キャスト) が行われるか?**
  OS/400 PASE では、パフォーマンスへの影響が少ない 32 ビットと 64 ビットの両方の AIX アドレッシング・モデルがサポートされており、整数をポインターに変換できます。

OS/400 PASE が必ずしも最善のソリューションではない場合

ILE から呼び出さなければならない、多数の呼び出し可能インターフェースを提供するコードや、以下のいずれかの特性を持つコードの場合は、一般に OS/400 PASE は適していません。

- 呼び出しごとに OS/400 PASE を開始または終了するか、またはすでにアクティブな OS/400 PASE プログラムで OS/400 PASE プロシージャーを呼び出す (Qp2CallPase API を使用して) かのいずれよりも、パフォーマンスの点で優れた呼び出しおよび戻りが必要なコード。
- ILE 呼び出し元とライブラリー・コードの間で、メモリーまたはネーム・スペースを共用する必要があるコード。 OS/400 PASE プログラムは、呼び出し元の ILE コードと暗黙的にメモリーおよびネーム・スペースを共用することはありません。 (ただし、OS/400 PASE から呼び出される ILE コードでは、OS/400 PASE メモリーを共用または使用できます。)

OS/400 PASE のインストール

OS/400 PASE は、すべての iSeries サーバーで、課金なしで利用できます。 OS/400 PASE のインストールは、推奨されています。拡張ドメイン・ネーム・サーバー (DNS) や ILE C++ コンパイラーなどの一部のシステム・ソフトウェアでは、 OS/400 PASE サポートが必要です。

サーバーに OS/400 PASE をインストールするには、次のようにします。

1. OS/400 コマンド行で、`GO LICPGM` と入力します。
2. 11 を選択します。ライセンス・プログラムをインストールします。
3. オプション 33 (5722SS1 - ポータブル・アプリケーション・ソリューション環境) を選択します。

ロケールのインストール

OS/400 PASE 製品では、 OS/400 にインストールされる言語フィーチャーに関連付けられたロケール・オブジェクトのみがインストールされます。サーバーの言語フィーチャーに組み込まれていないロケールが必要であれば、追加の OS/400 言語フィーチャーをオーダーしてインストールすることが必要な場合があります。詳細については、OS/400 PASE グローバリゼーションおよび OS/400 PASE ロケール (OS/400 PASE locales)を参照してください。 《

OS/400 PASE にアプリケーションを移植するソフトウェア開発者のための ライセンスの注:

OS/400 PASE には、OS/400 システム上に AIX ランタイム・ライブラリーのサブセットがあります。OS/400 に同梱されているライブラリー・コードはすべて、OS/400 のライセンスで使用できます。ただし、このライセンスは、 OS/400 PASE に同梱されていない AIX ライブラリーに対するライセンスを意味するものではありません。すべての AIX 製品のライセンスは、IBM によって個別に交付されます。

独自のアプリケーションを OS/400 PASE に移植しようとするとき、そのアプリケーションが、 OS/400 PASE に同梱されていない AIX ライブラリーに依存しているという場合があるかもしれません。そのような場合は、これらのライブラリーを OS/400 システムに移植する前に、それらのライブラリーがどのソフトウェア製品で提供されているのかを確認し、そのソフトウェア製品のライセンス許諾条件を調べる必要があります。場合によっては、IBM やサード・パーティーと連絡をとり、アプリケーションが依存する付加的なミドルウェアを OS/400 システムに移植する必要があります。移植を行うときは、それを開始する前に、移植しようとしているコードに関係するすべてのライセンス許諾条件をよく調べてください。 IBM に帰属すると思われるライブラリーに関してライセンス許諾条件の情報が必要な場合は、 IBM の営業担当員、いずれかの IBM ポーティング・センター、ロチェスターの Custom Technology Center、または PartnerWorld for Developers に相談してください。

OS/400 PASE の計画

OS/400 PASE は、OS/400 上に、最小限の手間で iSeries サーバーに AIX アプリケーションを移植できるようにする、AIX ランタイム環境を提供します。実際、多くの AIX プログラムは、変更を加えなくても OS/400 PASE で稼働します。これは、OS/400 PASE が、AIX 上で使用可能なものと同じ共用ライブラリーを数多く備えており、 pSeries(TM) AIX PowerPC プロセッサーで稼働するのと同じように直接 iSeries PowerPC プロセッサーで稼働する、広範な AIX ユーティリティーのサブセットを備えているからです。

OS/400 PASE での作業を開始するにあたっては、次のいくつかの点に留意してください。

- **» AIX バイナリーのターゲット・リリースと、バイナリーが稼働する OS/400 PASE のリリースとの間には、相互関係があります。**

  OS/400 PASE アプリケーションを AIX 上でコンパイルする場合、AIX で作成するアプリケーション・バイナリーには、そのアプリケーションが実行される OS/400 PASE のバージョンとの互換性が必要です。次の表は、OS/400 PASE の各バージョンと互換性がある AIX バイナリーのバージョンを示しています。たとえば、AIX リリース 4.3 用に作成される 32 ビット・アプリケーションは、 OS/400 PASE V4R5、V5R1、または V5R2 で稼働しますが、OS/400 PASE V4R4 では稼働しません。同様に、AIX リリース 4.3 用に作成される 64 ビット・アプリケーションは、 OS/400 PASE V5R1 で稼働しますが、OS/400 PASE V4R4、V4R5、および V5R2 では稼働しません。 《

AIX のリリース	OS/400 V4R4	OS/400 V4R5	OS/400 V5R1	OS/400 V5R2
4.2 (32 ビット)	X	X	X	X
4.3 (32 ビット)	-	X	X	X
4.3 (64 ビット)	-	-	X	-
5.1 (32 または 64 ビット)	-	-	-	X (注を参照)

» **注:** バージョン 5 リリース 2 (64 ビット) の OS/400 PASE アプリケーションは、 AIX 5L(TM) リリース 5.1 で再コンパイルする必要があります。詳細については、OS/400 PASE で実行するプログラムの準備を参照してください。 «

- **OS/400 PASE では、OS/400 上に AIX カーネルがありません。**

  代わりに、共用ライブラリーが必要とする低レベル・システム関数は、すべて、OS/400 カーネルか統合 OS/400 機能にルーティングされます。この点で、OS/400 PASE は、AIX プラットフォームと OS/400 プラットフォームとの間の架け橋となります。共用ライブラリー内の API では、AIX の場合と同じ構文が使用されますが、OS/400 PASE プログラムは OS/400 ジョブの中で実行され、他のすべての OS/400 ジョブとまったく同じように OS/400 によって管理されます。

- **ほとんどのケースにおいて、 OS/400 PASE で呼び出される API は、AIX 上とまったく同じように動作します。**

  ただし、一部の API は、OS/400 PASE では違う動作をするか、あるいは OS/400 PASE ではサポートされていません。このため、OS/400 PASE プログラムの準備の計画を立てる際は、 API 分析ツールを使用したコード全体の分析をまず行う必要があります。このツールを使用することにより、 AIX アプリケーションを OS/400 PASE に移植する際に考慮すべきプログラムの修正のタイプについて、総括的な概要を知ることができます。

- **AIX プラットフォームと OS/400 プラットフォームのいくつかの違いについても考慮してください。**
  - AIX には、一般に大文字と小文字の区別がありますが、ある OS/400 ファイル・システムには、この区別がありません。
  - AIX ではデータ・エンコードに ASCII を使用するのが一般的ですが、 OS/400 では通常 EBCDIC が使用されます。 OS/400 PASE プログラムからの統合言語環境 (ILE) の呼び出しの詳細を管理したい場合には、これは考慮事項となります。たとえば、OS/400 PASE から任意の ILE プロシージャーへの呼び出しを行う場合には、文字エンコード変換を処理するように、明示的に OS/400 PASE プログラムをコーディングする必要があります。 OS/400 PASE のランタイム・サポートには、文字エンコード変換のための `iconv_open()`、`iconv()`、および `iconv_close()` 関数が含まれています。

    注: `iconv()` インターフェースのインプリメントは、 OS/400 PASE と ILE で独立しており、それぞれ固有の変換テーブルがあります。 OS/400 PASE `iconv()` 関数でサポートされる変換は、統合ファイル・システムに保管されるバイト・ストリーム・ファイルとして保管されるため、ユーザーによって変更および拡張することができます。
  - AIX アプリケーションは、行 (ファイルやシェル・スクリプト内の行など) の終わりが LF 改行になっていることを予期します。しかし、パーソナル・コンピューター (PC) ソフトウェアや OS/400 ソフトウェアでは、通常は行の最後に CRLF 改行が使用されます。
  - AIX 上で使用される一部のスクリプトやプログラムは、標準のユーティリティーにハード・コーディングされたパスを使用する場合があるため、 OS/400 PASE で使用するパスを反映して、パスに変更を加える必要があります。詳細については、OS/400 PASE とのプログラムの互換性の分析を参照してください。

これらの問題のいくつかは、OS/400 PASE によって自動的に処理されます。たとえば、システムによって提供される OS/400 PASE ランタイム・サービス (OS/400 オプション 33 に同梱されている共用ライブラリーのシステム呼び出しやランタイム機能すべてを含む) を使用する場合、OS/400 PASE は、必要に応じて ASCII 対 EBCDIC の変換を実行します。ただし、一般的にファイル記述子 (バイト・ストリーム・ファイルまたはソケット) で読み書きされるデータに対しては、変換は実行されません。

ILE 関数や API への呼び出しを行う OS/400 PASE プログラムの機能を拡張する場合には、 _ILECALL などの他の低レベル関数を使用できます。しかし、前述のとおり、データ変換を操作する必要はありません。また、プログラムにこれらの拡張子をコーディングするには、付加的なヘッダーおよびエクスポート・ファイルの使用が必要になります。

OS/400 PASE で実行するプログラムの準備

OS/400 で効果的に稼働する AIX プログラムを準備するためのステップは、プログラムの性質や、OS/400 固有のインターフェースや機能を使用する必要があるかどうかによって変わります。

OS/400 PASE にUNIX アプリケーションを移植しようとする場合は、まずアプリケーションが » AIX コンパイラー « を使用してコンパイルすることを確認してください。場合によっては、この要件を満たすために UNIX プログラムを修正する必要があります。

» **注:** OS/400 PASE V5R2 は、 AIX 5L リリース 5.1 をサポートしています。 OS/400 PASE で実行したい新しい 64 ビット・アプリケーションは、 AIX 5L リリース 5.1 でコンパイルされなければならず、既存の 64 ビット・アプリケーションにも再コンパイルが必要です。 32 ビット・アプリケーションは、以前のリリースの AIX で使用されていたものも、再コンパイルなしで OS/400 PASE でサポートされます。 «

ステップ 1: プログラムの分析

このプロセスの最初のステップは、すべてのケースで推奨されます。 API 分析ツールを使用して、プログラムで使用する API についての詳細なレポートを入手し、それらが OS/400 PASE でどのように実行されることを予期できるかを確認してください。

ステップ 2: AIX ソース・プログラムのコンパイル

プログラムの OS/400 PASE プログラムとしての適正を確認した後、 OS/400 PASE で実行するために必要な何らかの変更を行った場合は、ソースをコンパイルします。 (AIX プログラムの分析によって、何も変更を加えなくても OS/400 PASE で実行できることが確認された場合は、プログラムを再コンパイルする必要はありません。)

» OS/400 PASE プログラムをコンパイルするのに、AIX システムを使用するか、または OS/400 PASE 環境でプログラムをコンパイルするために、OS/400 PASE での二つの AIX コンパイラー製品のうちの一つを任意にインストールすることができます。 «

ステップ 3: iSeries サーバーへのプログラムのコピー

» OS/400 PASE プログラムを AIX システムでコンパイルした場合は、バイナリー・ファイルを iSeries サーバーにコピーしてください。 «

ステップ 4: (オプション) OS/400 インターフェースを使用するための AIX アプリケーションのカスタマイズ

OS/400 固有のインターフェースを使用するために AIX アプリケーションをカスタマイズし、 » AIX 上でアプリケーションをコンパイルする場合は «、OS/400 PASE プログラムをコンパイルする前に、 1 つ以上の OS/400 ヘッダー・ファイルまたはエクスポート・ファイルを AIX システムにコピーする必要があります。

ご使用のプログラムと OS/400 PASE の互換性の分析

UNIX C アプリケーションの iSeries サーバーへの移植性を評価する最初のステップには、アプリケーションで使用されるインターフェースを分析することが関係しています。この API 分析により、アプリケーション内で使用されるインターフェースが業界標準でないことや、 OS/400 でサポートされていないことが明らかになります。また、インターフェースが標準に準拠しているとしても、 UNIX マシンとは OS/400 のアーキテクチャーが異なるため、別の方法でサポートされていることも確認できます。

API Analysis Tool は、フロントエンドおよびバックエンドのプロセスから成り、フロントエンド・プロセスでは、コンパイル済みのアプリケーションをスキャンして、アプリケーションで使用されるインターフェース (外部関数およびデータ) を抽出し、それらのすべてのインターフェースのリストを生成します。バックエンド・プロセスでは、このインターフェースのリストを入力として使用し、典型的なシステム API およびそれらのサポートから成るデータベースとインターフェースとを比較します。

API 分析ツールのフロントエンド・プロセスは UNIX シェル・スクリプトです。これは、`nm` または `dump` コマンドを使用して、アプリケーションの外部シンボル・テーブルからシンボル情報を見つけます。

シンボルからストリップされたバイナリーには、分析するツールに関する、動的バインディング情報が十分に含まれていることがあります。静的にバインドされたバイナリーでは、ライブラリー・インターフェースを分析の対象に含めませんが、システム呼び出しの依存関係は分析用に引き続き公開します。

コンパイル前に実行する追加の分析

API 分析ツールから収集する情報に加えて、以下の情報も収集する必要があります。

- **アプリケーションで使用されるライブラリーのリストの取得:** 分析ツールでは、アプリケーションで使用される標準 API の一部についてフィードバックを提供しますが、多数の共通 API セットを探すわけではありません。ライブラリー分析は、アプリケーションで使用されるミドルウェア API のいくつかを識別するのに役立ちます。ご使用のコマンドおよび共用オブジェクトのそれぞれに対して以下のコマンドを実行して、アプリケーションで必要なライブラリーのリストを入手することができます。

  ```
  dump -H binary_name
  ```

- **ハードコーディングされたパス名の検査:** クリデンシャルを変更するプログラムを実行する場合、または OS/400 PASE 環境変数が PASE_EXEC_QOPENSYS=N であるときでもコマンドまたはスクリプトを実行させる場合は、ハードコーディングされたパス名を変更しなければならないことがあります。

  `/usr/bin/ksh` は絶対パス (ルートで始まる) であるため、それが見つからない場合、またはそれがバイト・ストリーム・ファイルでない場合は、 OS/400 PASE は自動的に /QOpenSys ファイル・システムを検索して、パス名 `/QOpenSys/usr/bin/ksh` を探します。 QShell ユーティリティー・プログラムはバイト・ストリーム・ファイルではないため、オリジナルの (絶対) パスが QShell ユーティリティー・プログラムに対するシンボリック・リンク (`/usr/bin/sh` など) である場合でも、OS/400 PASE は /QOpenSys ファイル・システムを探します。

AIX ソースのコンパイル

プログラムが AIX インターフェースのみを使用する場合、必須の AIX ヘッダーとコンパイルし、AIX ライブラリーとリンクして、 OS/400 PASE 用のバイナリーを作成します。 OS/400 PASE は、 AIX システムが提供する共用ライブラリーと静的に結合するアプリケーションはサポートしません。

OS/400 PASE プログラムの構造と、 PowerPC 用の AIX プログラムの構造は同一です。

» OS/400 PASE Option 33 にはコンパイラーは含まれません。 OS/400 PASE プログラムをコンパイルするのにAIX システムを使用するかまたは、OS/400 PASE 環境でコンパイルするために、OS/400 PASE での二つの AIX コンパイラー製品のうちのひとつを任意にインストールすることができます。 «

pSeries サーバーでの AIX コンパイラーの使用

PowerPC 用の AIX Application Binary Interface (ABI) と互換性がある出力を生成する AIX コンパイラーおよびリンカーを使用して、 OS/400 PASE プログラムを作成することができます。 OS/400 PASE は、PowerPC には存在しない POWER アーキテクチャー指示 (キャッシュ管理 POWER 指示は存在する) を使用するバイナリーの指示エミュレーション・サポートを提供します。

OS/400 PASE での AIX コンパイラーの使用

» OS/400 PASE は、OS/400 PASE 環境で別々に使用可能な AIX コンパイラーのどちらかのインストールをサポートします。

- IBM VisualAge C++ Professional for AIX, バージョン 6 (5765-F56) (この製品は、 IBM C for AIX コンパイラーを含みます。)
- IBM C for AIX, バージョン 6 (5765-F57)

この製品を使用して開発、コンパイル、ビルドおよび iSeries サーバー上の OS/400 PASE 環境内で AIX アプリケーションを実行します。

この製品のオーダーおよびインストールについての詳細については、 Application Factory Web サイトの OS/400 PASE における VisualAge C++ for AIX コンパイラーのインストール  ページを参照してください。 «

開発ツール

» OS/400 PASE には、AIX で使用する多くの開発ツール (例: ld、ar、make、yacc) が付属しています。詳細は、OS/400 PASE シェルおよびユーティリティーのトピックを参照してください。 OS/400 PASE で使用できる、他のソースからの AIX ツール (たとえば、オープン・ソース・ツール gcc など) も多数あります。

また、iSeries Tools for Developers PRPQ (5799-PTL) にも、 iSeries アプリケーションを開発、構築、移植する上で役立つ多彩なツールが含まれています。この PRPQ については、 Web サイト Application Factory - iSeries Tools for Development  を参照してください。 «

ポインターの処理に関するコンパイラーの注意事項

- `xlc` コンパイラーは、 `-qlngdbl128` と `-qalign=natural` を組み合わせて使用することにより、 (長倍精度実数型の) 16 バイト調整の限定サポートを提供します。 ILEpointer 型は、MI ポインターが構造内で 16 バイトに調整されるようにするために、これらのコンパイラー・オプションを必要とします。オプション `-qldbl128` を使用すると、長倍精度実数型は強制的に 128 ビット型になります。この場合、長倍精度実数フィールドの `printf` のような操作を処理するために libc128.a を使用する必要があります。

  `xlc` コマンドの代わりに `xlc128` コマンドを使用すると、簡単にオプション `-qlngdbl128` を入手し、libc128.a とリンクすることができます。
- `xlc/xlC` コンパイラーには現在、静的変数または自動変数の 16 バイト調整を強制する手段がありません。このコンパイラーは、構造内の 128 ビット長倍精度実数フィールドの相対調整を保証するだけで

す。 malloc の OS/400 PASE バージョンは常に 16 バイト調整ストレージを提供するので、スタック・ストレージの 16 バイト調整を行うことができます。

- また、ヘッダー・ファイル as400_types.h も、 64 ビット整数である long long 型に依存します。 xlc コンパイラー・オプション -qlonglong は、この形状を保証します (これは、 xlc コンパイラーを実行する一部のコマンドではデフォルトになっていません)。

例

» 以下の例は、AIX システムで OS/400 PASE プログラムをコンパイルする際に使用するためのものです。プログラムをコンパイルするために、OS/400 PASE にインストールされたコンパイラーを使用すると、 OS/400 固有のヘッダー・ファイルまたは、OS/400 固有のエクスポート・ファイルの場所についてのコンパイラー・オプションを指定する必要はありません。それは、これらのファイルが OS/400 システム上で /usr/include/ および /usr/lib/ のデフォルト・パスの位置にあるためです。 «

例 1: 以下の AIX システム上のコマンドを使用すると、 *testpgm* という OS/400 PASE プログラムが作成されます。これは、libc.a によってエクスポートされる OS/400 固有のインターフェースを使用できます。

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

この例では、OS/400 固有のヘッダー・ファイルが AIX ディレクトリー /usr/include にコピーされ、OS/400 固有のエクスポート・ファイルが AIX ディレクトリー /mydir にコピーされることを前提としています。

例 2: 以下の例では、 OS/400 固有のヘッダー・ファイルおよびエクスポート・ファイルが /pase/lib にあることを前提としています。

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

例 3: 以下の例では、同じオプションを使用して、例 2 と同じプログラムを作成しています。ただし、xlc_r コマンドがマルチスレッド・プログラムで使用され、コンパイル済みアプリケーションがスレッド・セーフのランタイム・ライブラリーとリンクするようになっています。

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

この例では、DB2 UDB for iSeries 呼び出しレベル・インターフェース (CLI) 用の OS/400 PASE サポートを使用する場合、 build コマンドで -bI:/pase/include/libdb400.exp も指定する必要があります。

-bI ディレクティブは、パラメーターを ld コマンドに渡すようにコンパイラーに命令します。このディレクティブは、ライブラリーからエクスポートした記号を含むエクスポート・ファイルが、アプリケーションによってインポートされるように指定します。

iSeries サーバーへの OS/400 PASE プログラムのコピー

OS/400 PASE で実行する AIX バイナリーを統合ファイル・システムにコピーします。統合ファイル・システムで使用できるファイル・システムはすべて、 OS/400 PASE 内で使用することができます。統合ファイル・システムの詳細については、 統合ファイル・システムのトピックを参照してください。

プラットフォーム間でファイルを移動する場合、問題が生じる可能性のある以下の違いに注意してください。

- アプリケーションが大文字小文字の区別を行う場合、 /QOpenSys ファイル・システム、または大文字小文字を区別するものとして作成したユーザー定義のファイル・システムにファイルを移動します。
- AIX と OS/400 では、テキスト・ファイル内 (たとえばファイルおよびシェル・スクリプト内) で使用する改行文字が異なります。

ファイルの転送

iSeries サーバーとの間の OS/400 PASE プログラムおよび関連ファイルの転送は、以下のいずれかの方法で行うことができます。

- ファイル転送プロトコル (FTP)
- Server Message Block (SMB)
- リモート・ファイル・システム

ファイル転送プロトコル (FTP) を使用したファイルのコピー

OS/400 FTP デーモンおよびクライアントを使用することによって、 OS/400 統合ファイル・システムとの間でファイルの転送を行うことができます。ファイルの転送を行う際はバイナリー・モードを使用します。FTP サブコマンド `binary` を使用してこのモードを設定してください。

ファイルを統合ファイル・システムに配置する際には、命名形式 1 (OS/400 FTP コマンドの NAMEFMT 1 サブコマンド) を使用する必要があります。この形式により、UNIX パス名を使用することができ、ストリーム・ファイルにファイルを転送します。命名形式 1 を使用するには、以下のいずれかを行います。

- UNIX パス名を使ってディレクトリーを変更します。これで、セッションが自動的に命名形式 1 になります。この方法を使用すると、最初のディレクトリーの前にスラッシュ (/) が付けられます。たとえば、次のようになります。

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

- リモート・クライアントの場合は FTP サブコマンド `quote site namefmt 1` を使用し、ローカル・クライアントの場合は `namefmt 1` を使用します。

FTP の詳細については、 FTP のトピックを参照してください。

Server Message Block (SMB) を使用したプログラムのコピー

OS/400 は、SMB クライアント・コンポーネントおよびサーバー・コンポーネントをサポートします。NetServer を構成して実行すると、 OS/400 PASE は /QNTC ファイル・システムを使用してネットワーク内の SMB サーバーにアクセスできます。 UNIX プラットフォームでこれと同じサービスを提供するには、 SAMBA サーバーが必要となります。構成済みの操作可能 UNIX システム (AIX など) をインストールすると、 OS/400 PASE で使用可能なディレクトリーおよびファイルを作成することができます。

リモート・ファイル・システムを使用したプログラムのコピー

OS/400 では、統合ファイル・システムのファイル・スペース内のマウント・ポイントに、ネットワーク・ファイル・システム (NFS) をマウントすることができます。 AIX では、分散ファイル・システム (DFS)(TM) と Andrew File System (AFS)(R) に加えて NFS もサポートしており (DFS から NFS に、また AFS から NFS に変換するプログラムを使用)、 OS/400 はこれらのファイル・システムのエクスポートおよびマウントが行えます。これにより、OS/400 PASE もこれらのファイル・システムを使用することができます。 OS/400 ユーザー・プロファイルのユーザー ID 番号およびグループ ID 番号を使用して、アク

セス対象のディレクトリー・パスおよびファイルに対するセキュリティー権限が検証されます。複数のプラットフォームで同一のユーザーとなるように意図されたユーザー・プロファイルには、すべてのシステム上で同じユーザー ID が含まれるようにします。

OS/400 は NFS サーバーとして使用されるときに最もその性能を発揮します。 NFS サーバーとして使用する場合、 AIX システムから OS/400 統合ファイル・システムのディレクトリーにマウントすると、プログラムを作成する際に OS/400 に直接作成されます。

注: OS/400 NFS は現在、マルチスレッド・アプリケーションではサポートされていません。

大文字小文字の区別

UNIX システム・インターフェースでは一般に、大文字と小文字が区別されます。 OS/400 では、大文字小文字の区別がない場合もあります。特に、大文字小文字の区別があることによって、既存のコードとの混乱が生じる可能性があるいくつかの状況を把握しておく必要があります。

ディレクトリーまたはファイル単位の大文字小文字の区別は、 OS/400 で使用しているファイル・システムに依存します。 /QOpenSys ファイル・システムでは大文字小文字の区別があり、大文字小文字の区別があるユーザー定義のファイル・システム (UDFS) を作成することができます。さまざまなファイル・システムの特性の詳細は、 統合ファイル・システムのトピック (特に統合ファイル・システムのファイル・システムの比較のトピック) を参照してください。

また、OS/400 のユーザー ID とグループ ID も常に大文字で戻されることに注意してください。

例

以下に、大文字小文字の区別から生じる可能性のある問題の数例を示します。

例 1: この例では、シェルが `readdir()` による戻り値に対して総称名接頭部の文字比較を行います。ただし、QSYS.LIB ファイル・システムは大文字のディレクトリー項目を戻すため、どの項目も小文字の総称名接頭部とは一致しません。

```
$ 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
```

例 2: この例は最初の例と類似していますが、シェルではなく `find` ユーティリティーによって比較が行われている点が異なります。

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

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

例 3: `ps` ユーティリティーはユーザー名に大文字小文字の区別があることを予期しているため、 `-u` オプションによって指定された大文字の名前と OS/400 PASE ランタイム機能 `getpwuid()` によって戻される小文字の名前との一致は認識しません。

```
$ 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
```

統合ファイル・システムにおける改行文字

OS/400 PASE プログラムのソースの AIX アプリケーションでは、行 (ファイルやシェル・スクリプト内の行など) の終わりに LF 改行を入れることが求められています。ただし、PC ソフトウェアや一般的な OS/400 ソフトウェアでは、行の最後が CRLF 改行になっている場合も少なくありません。

FTP での CRLF の使用

この違いが問題の原因となる 1 つの例は、 FTP を使ってソース・ファイルとシェル・スクリプトを AIX から iSeries に転送するケースです。 FTP の標準では、テキスト・モードで送信され、行の最後に CRLF 改行が使用されたデータが必要です。一方 AIX では、テキスト・モードのインバウンド・ファイルを処理する際に、 FTP ユーティリティーによって CR 改行が除去されます。 OS/400 FTP は、必ずデータ・ストリームに示されているとおりに書き込みを行い、必ずテキスト・モード用に CRLF を残すため、これが OS/400 PASE のランタイムやユーティリティーでは問題の原因になります。

この問題を防ぐため、可能な場合は、UNIX システムからの転送にはバイナリー・モードを使用してください。また、パーソナル・コンピューターから転送されるテキスト・ファイルでも、ほとんどの場合に CRLF 区切り文字が使用されています。そのような場合は、ファイルをまず AIX に転送するようにすれば、問題を修正できます。現行ディレクトリー内のファイルから CR を除去する手段としては、次のような予備手段もあります。

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

iSeries や PC のエディターでの CRLF の使用

iSeries サーバーやワークステーションのエディター (Windows(R) Notepad エディターなど) でファイルやシェル・スクリプトを編集する場合にも問題は生じます。これらのエディターで使用される改行区切り文字は CRLF であって、OS/400 PASE で使用される LF ではないからです。

改行区切り文字として CRLF を用いないエディターは数多くあります (例: ez エディター)。 IBM PartnerWorld の Web サイトにある、 開発者のための各種 iSeries ツール のリストを参照してください。

OS/400 機能を使用するための OS/400 PASE プログラムのカスタマイズ

AIX アプリケーションで、システム提供の OS/400 PASE 共用ライブラリーでは直接サポートされていない OS/400 の機能を利用したい場合は、いくらかの付加的なステップを実行してアプリケーションを準備する必要があります。

まず、OS/400 固有の機能へのアクセスを調整する、すべての必要な OS/400 PASE ランタイム機能が呼び出されるように、 AIX アプリケーションをカスタマイズする必要があります。

次に、 »AIX システム上の OS/400 PASE プログラムをコンパイルする場合は、カスタマイズしたアプリケーションをコンパイルする前に、以下のステップを実行する必要があります。 «

- 必要な OS/400 固有のヘッダー・ファイルを AIX システムにコピーする
- 必要な OS/400 固有のエクスポート・ファイルを AIX システムにコピーする

OS/400 PASE と OS/400 機能の統合に関する詳細については、以下のトピックを参照してください。

- OS/400 PASE プログラムからの OS/400 プログラムおよび機能の呼び出し
- OS/400 PASE プログラムと OS/400 機能の相互作用

ヘッダー・ファイルのコピー

OS/400 PASE は、標準の AIX ランタイムに、OS/400 固有のサポート用のヘッダー・ファイルを追加します。これらのヘッダー・ファイルは、 OS/400 PASE および OS/400 オペレーティング・システムによって提供されます。これらのヘッダー・ファイルを、iSeries サーバーから AIX マシンのヘッダー・ファイル検索パスにコピーしてください。

ヘッダー・ファイルは、以下の AIX ディレクトリー、またはコンピューターのヘッダー・ファイル検索パスにある他の任意のディレクトリーにコピーできます。

```
/usr/include
```

/usr/include 以外のディレクトリーを使用する場合は、 AIX コンパイラー・コマンドの -I オプションを使用して、そのディレクトリーをヘッダー・ファイル検索パスに追加できます。

ファイルのコピーに関する詳細は、 iSeries サーバーへの OS/400 PASE プログラムのコピーを参照してください。

OS/400 PASE ヘッダー・ファイルのコピー

OS/400 PASE ヘッダー・ファイルは、以下の OS/400 ディレクトリーにあります。

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

OS/400 PASE では、次のようなヘッダー・ファイルが提供されています。

as400_protos.h	OS/400 PASE から ILE へ。各種の OS/400 固有の機能を提供します。
as400_types.h	ILE への呼び出し用の固有な OS/400 パラメーター・タイプ このヘッダー・ファイルは、16 バイト・マシン・インターフェース (MI) ポインターに、タイプ ILEpointer を宣言します。この宣言は、長倍精度実数型が 128 ビット・フィールドであることに依存します。 as400_types.h で宣言される他のタイプは、long long 型が 64 ビット整数であることに依存します。 as400_types.h で宣言されるタイプのサイズや位置合わせが適正なものとなるようにするため、 AIX コンパイラーは、オプション -qlngdbl128、-qalign=natural、および -qlonglong を指定して実行する必要があります。
os400msg.h	OS/400 メッセージの送受信を行うための関数

OS/400 ヘッダー・ファイルのコピー

OS/400 で提供されるヘッダー・ファイルは、以下のディレクトリーにあります。

```
/QIBM/include
```

アプリケーションで何らかの OS/400 API ヘッダー・ファイルが必要とされる場合は、まずヘッダー・ファイルを EBCDIC から ASCII に変換し、その変換したファイルを AIX ディレクトリーにコピーする必要があります。

EBCDIC のテキスト・ファイルを ASCII に変換する 1 つの方法として、 OS/400 PASE の Rfile ユーティリティーを使用できます。

次に示す例では、OS/400 PASE の Rfile ユーティリティーで OS/400 ヘッダー・ファイル /QIBM/include/qusec.h を読み取り、そのデータを OS/400 PASE CCSID に変換した後、各行から行末のブランクを除去して、その結果として生成されたものをバイト・ストリーム・ファイル ascii_qusec.h に書き込みます。

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

エクスポート・ファイルのコピー

iSeries サーバーから AIX ディレクトリーにエクスポート・ファイルをコピーします。

OS/400 固有の機能にアクセスする必要のあるアプリケーションを作成するには、以下の OS/400 ディレクトリーにあるエクスポート・ファイルを使用することをお勧めします。

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

これらのファイルを AIX ディレクトリーにコピーできます。 AIX システム上の共用ライブラリーにない記号を定義するには、 AIX ld コマンド (または compiler コマンド) で -bI: オプションを使用します。

OS/400 PASE は以下のエクスポート・ファイルを提供します。

as400_libc.exp	libc.a にある OS/400 固有の機能用のエクスポート・ファイル as400_libc.exp ファイルは、 libc.a の OS/400 PASE バージョンからのすべてのエクスポート (そのライブラリーの AIX バージョンによってエクスポートされないもの) を定義します。
libdb400.exp	OS/400 データベース機能用のエクスポート・ファイル libdb400.exp ファイルは、 (DB2 UDB for iSeries 呼び出しレベルインターフェース (CLI) がサポートする) OS/400 PASE libdb400.a ライブラリーからのエクスポートを定義します。

ファイルのコピーの詳細については、 iSeries サーバーへの OS/400 PASE プログラムのコピーを参照してください。

OS/400 機能にアクセスするための OS/400 PASE API

OS/400 PASE は、 ILE コードおよびその他の OS/400 関数にアクセスするためのいくつかの API を提供しています。どの API を使用するかは、コンパイラーをどの程度機能させるかではなく、どれだけの準備と構成を行うかに依存します。詳細は、OS/400 PASE API を参照してください。

OS/400 環境での OS/400 PASE プログラムの使用

OS/400 PASE プログラムは他の OS/400 プログラムを呼び出すことができ、他の OS/400 プログラムは OS/400 PASE プログラムのプロシージャーを呼び出すことができます。ご使用のコンピューティング環境へ OS/400 PASE プログラムを組み込む方法については、以下のトピックを参照してください。

OS/400 プログラムからの OS/400 PASE プログラムの実行

ジョブでの OS/400 PASE プログラムの開始、および ILE プログラムからの OS/PASE プロシージャーの呼び出しに関する情報と例を提供します。

OS/400 PASE プログラムからの OS/400 プログラムおよび機能の呼び出し

OS/400 PASE プログラムからの ILE プロシージャー、OS/400 プログラム、および CL コマンドの呼び出しに関する情報と例を提供します。

OS/400 PASE プログラムと OS/400 機能の相互作用
OS/400 PASE プログラムの使用方法および OS/400 機能との相互作用の方法に関する情報を提供します。

OS/400 PASE プログラムおよびプロシージャーの実行

OS/400 PASE では、いくつかの方法で OS/400 PASE プログラムを実行できます。

» QP2SHELL() および QP2SHELL2() «
呼び出し元のジョブの中で OS/400 PASE プログラムを実行する OS/400 プログラム。

QP2TERM()
対話式シェル環境で OS/400 PASE プログラムを実行する OS/400 プログラム。

Qp2RunPase()
ILE プロシージャー内から呼び出され、OS/400 PASE プログラムを開始および実行する、ILE プロシージャー。

Qp2CallPase()
ILE プロシージャー内から呼び出され、 OS/400 PASE 環境がすでに稼働しているジョブの中で OS/400 PASE プログラムを実行する、ILE プロシージャー。

環境変数の処理では、 OS/400 PASE 環境と OS/400 環境がどのように相互作用するかが説明されています。

OS/400 PASE プログラムでの作業を可能にする ILE プロシージャー

» OS/400 PASE には、 ILE コードが OS/400 PASE サービスに (OS/400 PASE プログラム内に特別なプログラミングすることなく) アクセスできるようにするための、いくつかの ILE プロシージャー API が用意されています。

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

詳細については、OS/400 PASE ILE プロシージャー API (OS/400 PASE ILE Procedure APIs) を参照してください。

ILE スレッドへの接続

OS/400 PASE で作成されていないスレッド (例: Java スレッドや ILE pthread_create で作成されたスレッド) で実行される ILE コードから、 OS/400 PASE プログラム内のプロシージャーを呼び出すことが可能です。 Qp2CallPase は、自動的に ILE スレッドを OS/400 PASE に (対応する OS/400 PASE pthread 構

造体を作成することによって) 接続させます。ただしこれは、OS/400 PASE プログラムの開始時に OS/400 PASE 環境変数 PASE_THREAD_ATTACH が Y に設定された場合のみです。

OS/400 PASE から OS/400 プログラムに結果を戻す

OS/400 _RETURN() 関数を使用すると、OS/400 PASE プログラムを呼び出し、 OS/400 PASE 環境を終了させることなく結果を戻すことが可能です。これによって、OS/400 PASE プログラムを開始し、QP2SHELL2 (QP2SHELL ではない) または Qp2RunPase API が戻された後、そのプログラムの中でプロシージャーを (Qp2CallPase を使用して) 呼び出すことが可能になります。 «

QP2SHELL() を使用した OS/400 PASE プログラムの実行

任意の OS/400 コマンド行から OS/400 PASE プログラムを実行する場合や、任意の高水準言語プログラム、バッチ・ジョブ、または対話式ジョブからプログラムを実行する場合は、「OS/400 PASE シェル・プログラムの実行」プログラム (QP2SHELL または » QP2SHELL2 «) を使用します。これらのプログラムは、呼び出し元のジョブの中で OS/400 PASE プログラムを実行します。プログラムでは、OS/400 PASE プログラムの名前がパラメーターとして渡されます。このプログラムの使用法に関する詳細については、QP2SHELL() および QP2SHELL2() の説明を参照してください。

QP2SHELL() プログラムは、新しい活動化グループで OS/400 PASE プログラムを実行します。 »
QP2SHELL2() プログラムは、呼び出し側の活動化グループで実行されます。 «

次の例では、OS/400 のコマンド行から ls コマンドを実行します。

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

»

変数を使用して QP2SHELL() に値を渡す

CL 変数を使用して QP2SHELL() に値を渡す場合、変数はヌル終了でなければなりません。たとえば、上のサンプルを次のような方法でコーディングする必要があるでしょう。

```
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() での OS/400 PASE プログラムの実行

QP2TERM() プログラムで OS/400 PASE 対話式端末セッションを開始します。以下のコマンドは、デフォルトの Korn シェル・プロンプト (/QOpenSys/usr/bin/sh) を画面に表示します。

```
call qp2term
```

このプロンプトから、 OS/400 PASE プログラムを個別のバッチ・ジョブとして実行します。 QP2TERM() は、対話式ジョブを使用して、ファイル stdin、stdout、および stderr に関する、出力の表示および入力の受け入れを行います。

Korn シェルがデフォルトですが、実行する任意の OS/400 PASE プログラムへ渡す任意の引き数ストリングの他に、そのプログラムのパス名を指定することもできます。

QP2TERM() で開始する対話式セッションからは、任意の OS/400 PASE プログラムおよび任意のユーティリティーを実行でき、 stdout および stderr が、端末の画面に表示およびスクロールされます。

OS/400 プログラム内からの OS/400 PASE プログラムの実行

OS/400 PASE プログラムを実行するには、Qp2RunPase() API を使用します。プログラム名、引き数ストリング、および環境変数を指定してください。 ILE プログラムで Qp2RunPase() API を使用する方法の詳細については、 Qp2RunPase() API の説明を参照してください。

Qp2RunPase() API は、呼び出し元のジョブの中で OS/400 PASE プログラムを実行します。これは OS/400 PASE プログラム (必要な共用ライブラリーをすべて含む) をロードし、プログラムに制御を渡します。

この API では、QP2SHELL() や QP2TERM() に比べ、より幅広く OS/400 PASE の実行方法を制御できます。

ILE プログラムの中でどのようにこの API を使用できるかの例は、 サンプル・プログラムを参照してください。

例: OS/400 プログラム内からの OS/400 PASE プログラムの実行: 次の ILE プログラム (特記事項を参照) では、OS/400 PASE プログラムを呼び出します。このサンプルは、このプログラムが呼び出す OS/400 PASE コードの例を示しています。

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

/* include file for QP2RunPase(). */

#include <qp2user.h>

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

void main(int argc, char*argv[])
 {
  /* Path name of PASE program */
  char *PasePath = "/home/samplePASE";
  /* Return code from QP2RunPase() */
  int rc;
  /* The parameter to be passed to the
     OS/400 PASE program */
  char *PASE_parm = "My Parm";
  /* Argument list for OS/400 PASE program,
```

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

上の ILE プログラムにより、次の OS/400 PASE プログラム (特記事項を参照) が呼び出されます。

```
#include <stdio.h>

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

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

     return 0;
}
```

OS/400 プログラム内からの OS/400 PASE プロシージャーの呼び出し

最初に Qp2RunPase() API が、ジョブの中で OS/400 PASE プログラムを開始および実行します。そのジョブで OS/400 PASE がすでにアクティブになっている場合は、エラーが戻されます。

» OS/400 PASE プログラムがすでに実行されているジョブの中で OS/400 PASE プロシージャーを呼び出すには、 Qp2CallPase() API および Qp2CallPase2() API を使用します。

Qp2CallPase() API の使い方の例は、 例: OS/400 プログラム内からの OS/400 PASE プロシージャーの呼び出しを参照してください。 «

例: OS/400 プログラム内からの OS/400 PASE プロシージャーの呼び出し: 次の ILE プログラム (特記事項を参照) では、 OS/400 PASE プロシージャーを呼び出します。

```
#include <stdio.h>
#include <qp2shell2.h>
#include <qp2user.h>
```

```
#define JOB_CCSID 0

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

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

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

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

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

環境変数の処理

OS/400 PASE 環境変数は ILE 環境変数に依存しません。ある環境で変数を設定しても、別の環境には影響を与えません。ただし、 OS/400 PASE プログラムを実行するために使用する方法に応じて、 OS/400 PASE から ILE に変数をコピーすることができます。

対話式 OS/400 PASE セッションの環境変数

ILE 環境変数は、 QP2SHELL() および QP2TERM() を使用して開始される場合にのみ、 OS/400 PASE に渡されます。 OS/400 PASE を開始する前に、環境変数の処理 (WRKENVVAR) コマンドを使用して、環境変数の変更、追加、または削除を行います。

呼び出し先 OS/400 PASE セッションの環境変数

OS/400 PASE が (Qp2RunPase() API を使用した) プログラム呼び出しから開始される場合、環境変数に対する完全制御が与えられます。 OS/400 PASE プログラムの呼び出し元 ILE 環境と関係のない環境変数を渡すことができます。

» CL コマンドを実行する前に ILE に環境変数をコピーする

systemCL ランタイム機能にオプションを指定して、 CL コマンドを実行する前に、 ILE 環境に OS/400 PASE 環境変数をコピーします。これは、OS/400 PASE `system` ユーティリティーのデフォルトの動作でもあります。 «

詳細については、 OS/400 PASE 環境変数 (OS/400 PASE Environment Variables) のトピックを参照してください。

OS/400 PASE プログラムからの OS/400 プログラムおよびプロシージャーの呼び出し

OS/400 PASE では、ILE プロシージャー、Java プログラム、OPM プログラム、OS/400 API、および OS/400 機能への統合アクセスを持つ CL コマンドを呼び出すためのメソッドを提供します。

以下のトピックは、OS/400 PASE 環境から呼び出しを行うための説明および例を示しています。

ILE プロシージャーの呼び出し
ILE プロシージャーを OS/400 PASE から呼び出すには、まず OS/400 PASE プログラムからの呼び出しを処理するように ILE プロシージャーをセットアップする必要があります。また、コンパイル済み AIX プログラムでプログラム変数および構造体をセットアップする必要もあります。

» OS/400 プログラムの呼び出し
OS/400 プログラムは OS/400 PASE プログラム内から呼び出すことができます。 «

OS/400 コマンドの実行
CL コマンドは OS/400 PASE プログラム内から実行することができます。

OS/400 プログラムおよびプロシージャーの一般構成要件

» OS/400 PASE プログラム環境から OS/400 環境に呼び出しを行う場合、一般に、OS/400 プログラムが活動化グループの *CALLER でコンパイルされていることを確認する必要があります。それには、以下の理由があります。

- 活動化グループ内で実行する、OS/400 PASE (Qp2RunPase API によって呼び出される) を開始したコードだけが、 Qp2CallPase などの ILE API を使用して、OS/400 PASE プログラムと対話できる。
- ILE ランタイムがマルチスレッド・ジョブ内の活動化グループを破壊する場合 (さらに、OS/400 PASE `fork` によって作成されたすべてのジョブがマルチスレッド対応の場合)、 ILE ランタイムはジョブ全体を終了することがある (OS/400 を終了することもある)。 ACTGRP(*CALLER) を使用すると、ジョブを終了しようとする前に、ジョブが終了してしまうことを避けられます。 «

`systemCL` ランタイムを使用して、 CL コマンド (CALL コマンドを含む) をマルチスレッド対応でない別個のジョブで実行すると、マルチスレッド対応のジョブで実行するという問題を避けることができます。

ILE プロシージャーの呼び出し

OS/400 PASE プログラムから ILE プロシージャーを呼び出すには、コード内で以下の API 呼び出しを行います。

1. OS/400 PASE を開始したプロシージャーに関連する ILE 活動化グループに、結合プログラムをロードします。これを行うには、_ILELOAD() API を使用します。 » OS/400 PASE を開始した活動化グループで結合プログラムがすでにアクティブになっている場合、このステップは不要になる場合があります。この場合、_ILESYM のステップに進むことができます。活動化マーク・パラメーターにゼロを指定し、現行の活動化グループのすべてのアクティブ結合プログラムに含まれるすべての記号を検索します。 «
2. ILE 結合プログラムを活動化するときにエクスポート済み記号を検索し、記号のデータまたはプロシージャーに 16 バイトのタグ付きポインターを戻します。これを行うには、_ILESYM() API を使用します。
3. ILE プロシージャーを呼び出して、 OS/400 PASE プログラムから ILE プロシージャーに制御を転送します。これを行うには、_ILECALL() API または _ILECALLX() API を使用します。

さらに、OS/400 PASE プログラムから ILE プロシージャーを呼び出す際に、以下のタスクを実行する必要があります。

- テラスペース用に ILE プロシージャーを使用可能にする
- テキストを適切な CCSID に変換する
- 変数と構造をセットアップする

テラスペース用に ILE プロシージャーを使用可能にする

OS/400 PASE から呼び出す ILE モジュールをコンパイルする際に、常にテラスペース・オプションを *YES に設定する必要があります。この設定を行わないと、 OS/400 PASE アプリケーションのジョブ・ログに MCH4433 エラー・メッセージ (ターゲット・プログラム &2 のストレージ・モデルは無効です (Invalid storage model for target program &2)) が記録されます。詳細については、 ILE 概念  を参照してください。

テキストを適切な CCSID に変換する

ILE と OS/400 PASE の間で渡されるテキストは、事前に適切な CCSID に変換しておかなければならない場合があります。この変換を行わないと、文字変数の中に判読できない値が入ってしまいます。

変数と構造をセットアップする

OS/400 PASE プログラムから ILE を呼び出すには、変数と構造をセットアップする必要があります。必要なヘッダー・ファイルを AIX システムに確実にコピーし、シグニチャー、結果タイプ、および引き数リスト変数をセットアップしなければなりません。

- **ヘッダー・ファイル:** ILE を呼び出すには、 OS/400 PASE プログラムにヘッダー・ファイル as400_types.h および as400_protos.h が含まれていなければなりません。 as400_type.h ヘッダー・ファイルには、 OS/400 固有のインターフェースで使用されるタイプの定義が含まれています。
- **シグニチャー:** シグニチャー構造には、順序の説明と、OS/400 PASE と ILE の間で渡される引き数のタイプが含まれます。呼び出そうとしている ILE プロシージャーによって指示されるタイプのエンコードは、 as400_types.h header ファイルにあります。シグニチャーに 4 バイト以下の固定小数点引き数、または 8 バイト以下の浮動小数点引き数が含まれる場合、次のプラグマを使用して、ILE C コードをコンパイルする必要があります。

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

  このプラグマを使用しない場合、 ILE の標準 C リンケージで、1 バイトまたは 2 バイトの整数引き数を 4 バイトに、 4 バイトの浮動小数点引き数を 8 バイトに拡張する必要があります。

- **結果タイプ:** 結果タイプは C の戻りタイプの動作と類似しており、複雑ではありません。
- **引き数リスト:** » 引き数リストは正しい順序のフィールドを持つ構造でなければなりません。そのタイプはシグニチャー配列のエントリーによって指定されます。 « size_ILEarglist() API および build_ILEarglist() API を使用することにより、シグニチャーに基づいて引き数リストを自動的に作成することができます。

OS/400 PASE から ILE プロシージャーを呼び出すためのプロセスを説明した例については、 例: ILE プロシージャーの呼び出しを参照してください。

例: ILE プロシージャーの呼び出し: 以下のコード例 (特記事項を参照) は、サービス・プログラムの一部である ILE プロシージャーを呼び出すための OS/400 PASE コードと、 プログラムを作成するためのコンパイラー・コマンドを示しています。この例には、2 つの UNIX プロシージャーが含まれています。 ILE プロシージャーの処理方法は異なりますが、どちらも同じ ILE プロシージャーを呼び出します。最初のプロシージャーは、OS/400 PASE が提供するメソッドを使用した、 _ILECALL API のデータ構造の構築を示しています。 2 番目のプロシージャーは、手動による引き数リストの作成を示しています。

OS/400 PASE C コード

以下の例には、コードを説明するコメントが含まれています。例を入力したり検討したりする際には、これらのコメントを必ずお読みください。

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

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

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

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

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

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

/*
```

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

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

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

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

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

   /*
    * Save the current PID in static storage so we
    * can determine when to re-initialize (after fork)
    */
   init_pid = getpid();
}

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

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

```
    ARG_UINT8,      /* requires #pragma nowiden in ILE code */
    sizeof(aggregate),
    ARG_INT16,
    ARG_END
};

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

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

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

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

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

/*
 * wrapper_2 accepts the same arguments and returns
 * the same result as the ILE procedure it calls. This
 * method uses a customized or declared structure for the
 * ILE argument list to improve execution efficiency and
 * avoid heap storage leaks if an exception or signal occurs
 */
int wrapper_2(int arg1, void *arg2, double arg3,
                        char arg4, aggregate arg5, short arg6)
{
    /*
     * xlc does not guarantee 16-byte alignment for
     * automatic (stack) variables of any type, so we
     * find an aligned area in an oversized buffer
     */
    char ILEarglist_buf[sizeof(ILEarglistSt) + 15];
    ILEarglistSt *ILEarglist = (ILEarglistSt*)ROUND_QUAD(ILEarglist_buf);
    /*
     * Assignment statements are faster than calling
     * build_ILEarglist()
     */
    ILEarglist->arg1 = arg1;
    ILEarglist->arg2.s.addr = (address64_t)arg2;
    ILEarglist->arg3 = arg3;
    ILEarglist->arg4 = arg4;
    ILEarglist->arg5 = arg5;
    ILEarglist->arg6 = arg6;
    /*
     * Use a saved PID value to check if the ILE pointer
     * is set. ILE procedure pointers inherited by the
     * child process of a fork() are not usable because
     * they point to an ILE activation group in the parent
     * process
     */
    if (getpid() != init_pid)
    do_init();
    /*
     * _ILECALL calls the ILE procedure. The stack may
     * be unwound, but no heap storage is orphaned if
     * an exception or signal occurs
     */
    _ILECALL(ILEtarget,
        &ILEarglist->base,
        signature,
        result_type);
```

```
    if (ILEarglist->base.result.s_int32.r_int32 == 1)
      printf("The results of best_wrapper function is: %s¥n", arg2);
    else if ( ILEarglist->base.result.s_int32.r_int32 == 0)
    printf("ILE received other than 1 or 2 for version.¥n");
    else printf("The db file never opened.¥n");
    return ILEarglist->base.result.s_int32.r_int32;
    }
    void main () {
      int version,
                  result2;
      char dbText[ 25 ];
      double dblNumber = 5.999;
      char justChar = 'a';
      short shrtNumber = 3;
      aggregate agg;
      strcpy( dbText, "none" );

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

ILE C コード

ここでは、OS/400 システムでこの例の ILE C コードを作成する方法が示されます。コードの作成先のライブラリーにはソース物理ファイルが必要です。この ILE の例にもコメントが含まれています。これらのコメントは、コードを理解する上で重要です。ソースを入力したり検討したりする際には、これらのコメントを検討する必要があります。

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

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

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

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

```
size_t       srcLen;
size_t       tgtLen;
int          result;

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

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

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

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

プログラムを作成するためのコンパイラー・コマンド

OS/400 PASE プログラムをコンパイルする際に、コンパイラー・オプション `-qalign=natural` および `-qldbl128` を使用して、長倍精度実数型の相対 16 バイト調整を強制しなければなりません。これは、ILEpointer 型の中で使用されます。この調整は OS/400 の ILE で必要です。オプション `-bI:` を使用する場合は、 as400_libc.exp の保管先パス名を入力する必要があります。

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

ILE C モジュールとサービス・プログラムをコンパイルする際には、テラスペース・オプションを使用します。このオプションを使用しないと、 OS/400 PASE は ILE C モジュールやサービス・プログラムとの対話が行えません。

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

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

最後に、DDS をコンパイルして、少なくとも 1 つのデータ・レコードを伝搬する必要があります。

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

OS/400 PASE からの OS/400 プログラムの呼び出し

OS/400 PASE アプリケーションを作成する際に、既存の OS/400 プログラム (*PGM オブジェクト) を利用することができます。

- CL CALL コマンドを実行するには、systemCL 関数を使用します。詳細および例については、 OS/400 PASE からの OS/400 PASE コマンドの実行を参照してください。
- OS/400 PASE プログラム内から OS/400 プログラムを呼び出すには、 _PGMCALL ランタイム機能を使用します。このメソッドの実行処理速度は systemCL ランタイム機能より高速ですが、文字ストリング引き数の自動変換は実行せず、異なるジョブのプログラムを呼び出すための機能も備わっていません。

_PGMCALL ランタイム機能を使用して OS/400 PASE プログラムのコマンドを呼び出す方法の例については、 例: OS/400 PASE からの OS/400 プログラムの呼び出しを参照してください。 «

例: OS/400 PASE からの OS/400 プログラムの呼び出し: 以下の例 (特記事項を参照) は、_PGMCALL ランタイム機能を使用して OS/400 PASE プログラムからプログラムを呼び出す方法を示しています。

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

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

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

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

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

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

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

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

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

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

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

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


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

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

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

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

   /* initialize the QSZRTVPR returned info structure */
```

```
   memset(rcvr, 0, sizeof(rcvr));

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

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

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

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

   return(0);
}
```

OS/400 PASE からの OS/400 コマンドの実行

OS/400 機能を使用する制御言語 (CL) コマンドを実行することにより、 OS/400 PASE プログラムの機能を拡張することができます。 systemCL ランタイム機能を使用して、 OS/400 PASE プログラム内から OS/400 コマンドを実行します。


OS/400 PASE から OS/400 コマンドを実行する場合、 systemCL ランタイム機能は ASCII から EBCDIC への文字ストリングの引き数の変換を自動的に処理して、別のジョブでプログラムを呼び出せるようにします。

OS/400 PASE プログラムで CL コマンドを実行する方法の例については、 例: OS/400 PASE からの OS/400 コマンドの実行を参照してください。

例: OS/400 PASE からの OS/400 コマンドの実行: 以下の例 (特記事項を参照) は、 OS/400 PASE プログラムからコマンドを呼び出す方法を示しています。

```
/* sampleCL.c

   example to demonstrate use of sampleCL to run a CL command

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

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

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

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

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

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

OS/400 PASE プログラムと OS/400 の相互作用

OS/400 の機能を使用するように OS/400 PASE プログラムをカスタマイズする場合は、プログラムがそれらの機能とどのように相互作用するかを考慮する必要があります。以下のトピックでは、基本的なガイダンスとなる情報や、 iSeries Information Center 内の詳細な OS/400 システム情報へのリンクが紹介されています。

- 通信
- データベース
- データ・エンコード
- ファイル・システム
- グローバリゼーション
- メッセージ・サービス
- 印刷
- 疑似端末 (PTY)
- セキュリティー
- 実行管理機能

通信

» OS/400 PASE は、ソケット通信用に AIX と同じ構文をサポートします。これは、細部にわたって他の UNIX(TM) と一致しない場合があります。

OS/400 PASE ソケット・サポートは AIX のソケット・インプリメンテーションと比較できますが、 OS/400 PASE は (AIX カーネルのソケット・インプリメンテーションの代わりに) OS/400 のソケット・インプリメンテーションを使用するため、 AIX の動作とは多少異なります。

OS/400 のソケット・インプリメンテーションは、 UNIX 98 ソケットとバークレー・ソフトウェア・ディストリビューション (BSD) ソケットの両方をサポートします。ほとんどの場合、OS/400 PASE は AIX インプリメンテーションの動作を取り入れることによって、これらのスタイルの違いを解決します。

加えて、実行中のアプリケーションのユーザー・プロファイルには、ソケット API でレベル・パラメーターを IPPROTO_IP に、 option_value パラメーターを IP_OPTIONS に指定するための *IOSYSCFG 特殊権限がなければなりません。 OS/400 でのソケットの使用の詳細は、 ソケット・プログラミングのトピックを参照してください。特に、バークレー・ソフトウェア・ディストリビューション (BSD) との互換性と、UNIX 98 互換性のトピックを参照してください。 «

データベース

OS/400 PASE は、DB2 UDB for iSeries 呼び出しレベル・インターフェース (CLI) をサポートしています。 AIX および OS/400 上の DB2 CLI は互いに適切なサブセットではないので、いくつかのインターフェースで多少の違いがあり、あるインプリメンテーションで存在する API が、別のインプリメンテーションでは存在しない場合もあります。そのため、以下の点を考慮する必要があります。

- AIX 上でコードの生成は行えるが、テストができない。そのため、AIX ではなく、 OS/400 PASE 内の別のプラットフォームでコードをテストしなければならない。
- » ヘッダー・ファイル sqlcli.h の OS/400 バージョンを使用してコンパイルしなければならない。このヘッダー・ファイルの AIX バージョンを使用してコンパイルしたプログラムは OS/400 PASE で実行できない。 «

OS/400 のデフォルトのエンコード・システムが EBCDIC であるのに対して、 AIX は ASCII を基にします。この違いのために、 OS/400 データベース (DB2 UDB for iSeries) と OS/400 PASE アプリケーションとの間のデータ変換が必要になる場合があります。

DB2 CLI が OS/400 PASE をインプリメントする際に、 OS/400 PASE が提供するライブラリー・ルーチンによって、文字データは自動的に ASCII から EBCDIC に、あるいは EBCDIC から ASCII に変換されます。この変換は、アクセスされるデータのタグ付き CCSID、および OS/400 PASE プログラムが実行されている ASCII CCSID に基づいて行われます。データベースがタグ付けされる、つまり CCSID 65535 を使用してタグ付けされる場合、自動変換は行われません。これはデータのエンコード形式を識別するため、また必要な変換を実行するために、アプリケーションに残されます。

CCSID の処理

Qp2RunPase() API を使用する場合は、 OS/400 PASE CCSID を明示的に指定する必要があります。

» OS/400 PASE CCSID の制御は、 API プログラム QP2TERM、QP2SHELL、または QP2SHELL2 を呼び出す前に、 ILE 環境で以下の変数を設定することによって行えます。

- PASE_LANG
- QIBM_PASE_CCSID

ILE 環境でこれらの変数の一方またはその両方が省略される場合、 QP2TERM、QP2SHELL、および QP2SHELL2 はデフォルトで、 OS/400 PASE CCSID および OS/400 PASE 環境変数 LANG を、ジョブの言語および CCSID 属性の OS/400 PASE に相当する最適なものを使用して設定します。 «

詳細については、 QP2TERM() および QP2SHELL() プログラムの説明を参照してください。

» libc.a の拡張によって、 OS/400 PASE アプリケーションは _SETCCSID() 関数を使用して、実行中のアプリケーションの CCSID を変更することができるようになります。

アプリケーションの CCSID を変更せずに、 OS/400 PASE アプリケーションが DB2 CLI 内部変換をオーバーライドできるようにする拡張もあります。 SQLOverrideCCSID400() 関数は、オーバーライド CCSID の整数を、1 つのパラメーターとして受け取ります。 «

注: CCSID オーバーライド関数 SQLOverrideCCSID400() を有効にするには、この関数を他のすべての SQLx() API の前に呼び出す必要があります。そうしない場合、要求は無視されます。

OS/400 PASE プログラムでの DB2 UDB for iSeries CLI の使用

OS/400 PASE プログラムで DB2 CLI を使用するには、ソースをコンパイルする前に、 sqlcli.h ヘッダー・ファイルと libdb400.exp エクスポート・ファイルを AIX システムにコピーする必要があります。DB2 CLI ライブラリー・ルーチンは、OS/400 PASE 環境の libdb400.a にあり、 pthread インターフェースを使用してインプリメントされます。これはスレッド・セーフティーを提供します。 » OS/400 PASE CLI 関数は多くの場合、対応する ILE CLI 関数を呼び出して、目的の操作を実行します。 «

DB2 UDB コール・レベル・インターフェースの詳細については、 DB2 UDB for iSeries SQL 呼び出しレベル・インターフェース (ODBC) のトピックを参照してください。

DB2 UDB for iSeries SQL 呼び出しレベル・インターフェースを使用して、 OS/400 PASE が DB2 UDB for iSeries にアクセスする方法の例については、 例: OS/400 PASE プログラムの DB2 UDB for iSeries CLI 関数の呼び出しを参照してください。

例: OS/400 PASE プログラムでの DB2 UDB for iSeries CLI 関数の呼び出し: 以下の例 (特記事項を参照) は、 DB2 UDB for iSeries SQL コール・レベル・インターフェースを使用して DB2 UDB for iSeries にアクセスする OS/400 PASE プログラムを示しています。

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

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

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

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

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

```
void fun_PrintError( SQLHSTMT );

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

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

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

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

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

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

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

      nml_ReturnCode = SQLAllocConnect( nml_HandleToEnvironment,
                                        &nml_HandleToDatabaseConnection );

      if ( nml_ReturnCode != SQL_SUCCESS ) {
         printf( "%s: SQLAllocConnect¥n", pszId );
```

```
        fun_PrintError( SQL_NULL_HSTMT );
        nml_ReturnCode = fun_ReleaseEnvHandle();
        printf( "%s: Terminating¥n", pszId );
        return SQL_ERROR;
    } else {
        printf( "%s: SQLAllocConnect() succeeded¥n", pszId );
    } /* endif */

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

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

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

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

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

    Nmi_vParam = SQL_TRUE;
    nml_ReturnCode = SQLSetStmtOption( nml_HandleToSqlStatement,
                                       SQL_ATTR_CURSOR_SCROLLABLE,
                                       ( SQLINTEGER * ) &Nmi_vParam );
```

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

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

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

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

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

```
      printf( "%s: SQLBindCol() succeeded¥n", pszId );
  } /* endif */

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

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

    return SQL_SUCCESS;
} /* end fun_Process */

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

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

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

     return nml_ReturnCode;
} /* end fun_DisConnect */

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

     nml_ReturnCode = SQLFreeEnv( nml_HandleToEnvironment );
     if ( nml_ReturnCode != SQL_SUCCESS ) {
        printf( "%s: SQLFreeEnv() failed¥n", pszId );
        fun_PrintError( SQL_NULL_HSTMT );
```

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

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

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

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

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

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

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

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

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

     printf( "%s: SqlState - %s¥n", pszId, chs_SqlState );
```

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

データ・エンコード

ほとんどの UNIX システムでは、ASCII 文字エンコードが使用されます。 » ほとんどの OS/400 機能では、EBCDIC 文字エンコードが使用されます。いくつかの OS/400 オブジェクト・タイプに対して CCSID (Coded Character Set Identifier) の値を指定することにより、オブジェクト内の文字データに関する特定のエンコードを識別することができます。 «

» OS/400 PASE バイト・ストリーム・ファイルには、CCSID 属性があります。 OS/400 PASE 外にあるほとんどのシステム・インターフェースはこの属性を使用して、ファイルから読み取られるテキスト・データ、およびファイルに書き込まれるテキスト・データを必要に応じて変換します。 OS/400 PASE は、ストリーム・ファイルから読み取られるデータ、およびストリーム・ファイルに書き込まれるデータの CCSID 変換を行いませんが (AIX と整合性があります)、 OS/400 PASE プログラムによって作成されたバイト・ストリーム・ファイルの CCSID 属性を、現行の OS/400 PASE CCSID 値に設定して、その他のシステムの関数がファイル内の ASCII テキストを適切に処理できるようにします。 «

OS/400 PASE 共用ライブラリーに付属する AIX API を使用する場合、 OS/400 PASE はほとんどのデータ変換を処理します。 » OS/400 PASE プログラムは、 OS/400 PASE ランタイムによって自動的に処理されないすべての文字データ変換について、共用ライブラリー `libiconv.a` で提供されている `iconv` 関数を使用することができます。たとえば、OS/400 PASE アプリケーションは一般に、 (_ILECALLX または _PGMCALL のいずれかを使用して) OS/400 API 関数を呼び出す前に、文字ストリングを EBCDIC に変換する必要があります。 «

ファイル・システム

OS/400 PASE プログラムは、統合ファイル・システムを介してアクセス可能なすべてのファイルとリソースにアクセスできます。これには、QSYS.LIB および QOPT ファイル・システム内のオブジェクトも含まれます。

バッファーに入れられる入出力

外部装置との入出力は、OS/400 上でバッファーに入れられます。これはデータ・ブロックを扱う入出力プロセッサーによって処理されます。一方、UNIX システムは、文字単位の (バッファーに入れられない) 入出力を操作します。 OS/400 では、特定の入出力シグナル (例: Enter キー、ファンクション・キー、システム要求) が、システムに割り込みを送信します。

データ変換サポート

OS/400 PASE プログラムは ASCII (または UTF-8) パス名を `open` 関数に渡して、バイト・ストリーム・ファイルをオープンします。この場合、名前は OS/400 によって使用されるエンコード・スキーマに自動的に変換されますが、オープン・ファイルから読み書きされるデータは変換されません。

データ変換についての詳細は、 データ・エンコードを参照してください。

ファイル記述子の使用

OS/400 PASE ランタイムは通常、ファイル `stdin`、`stdout`、および `stderr` 用の ILE C ランタイム・サポートを使用します。これらのファイルは、OS/400 PASE および ILE プログラムに対して、一貫した動作を提供します。

OS/400 PASE および ILE C は、標準入出力について同じストリームを使用します (stdin、stdout、および stderr)。 OS/400 PASE は常に、ファイル記述子 0、1、および 2 を使用して、標準入出力にアクセスします。しかし、ILE C は常に stdin、stdout、および stderr の統合ファイル記述子を使用するとは限らないので、 OS/400 PASE は OS/400 PASE ファイル記述子と統合ファイル・システム内の記述子との間のマッピングを提供します。このマッピングにより、 OS/400 PASE プログラムと ILE C プログラムは異なる記述子番号を使用して、同じオープン・ファイルにアクセスすることができます。

fcntl 関数の OS/400 PASE 拡張版である F_MAP_XPFFD を使用して、 OS/400 PASE 記述子を ILE 番号に割り当てることができます。 » これは、OS/400 PASE によって作成されなかった ILE 記述子のファイル操作を OS/400 PASE アプリケーションが行わなければならない場合に役立ちます。 «

fstatx 関数の OS/400 固有の拡張版である STX_XPFFD_PASE を使用すると、 OS/400 PASE プログラムは OS/400 PASE ファイル記述子用の統合ファイル・システム記述子番号を決定することができます。ファイル stdin、stdout、および stderr 用の ILE C ランタイム・サポートに付加されたすべての OS/400 PASE 記述子には特殊値 (負の数) が戻されます。

Qp2RunPase() API が呼び出されるときに ILE 環境変数 QIBM_USE_DESCRIPTOR_STDIO が Y または I に設定されている場合、 OS/400 PASE はファイル記述子 0、1、および 2 と統合ファイル・システムとの同期をとり、 OS/400 PASE プログラムと ILE C プログラムの両方で、ファイル stdin、stdout、および stderr に同じ記述子番号が使用されるようにします。このモードの作動中に OS/400 PASE コードまたは ILE C コードが記述子 0、1、および 2 をクローズまたは再オープンする場合、その変更はどちらの環境の stdin、stdout、および stderr の処理にも影響を与えます。

OS/400 PASE ランタイムは一般に、 OS/400 PASE ファイル記述子 (ソケットを含む) によって読み書きされるデータの文字エンコード変換を行いません。ただし、ILE C stdin から読み取られるデータ、または ILE C stdout および stderr に書き込まれるデータについて、 (OS/400 PASE CCSID とジョブ・デフォルト CCSID の間の) ASCII から EBCDIC への変換は行われます。

stdin、stdout、および stderr の自動変換は、 2 つの環境変数によって制御されます。

- 一般に適用される変数は、QIBM_USE_DESCRIPTOR_STDIO です。この変数を Y に設定すると、 ILE ランタイムはこれらのファイルに対してファイル記述子 0、1、または 2 を使用します。
- PASE 固有の環境変数は、QIBM_PASE_DESCRIPTOR_STDIO です。バイナリーの場合は B、テキストの場合は T の値が入ります。

OS/400 PASE stdin、stdout、および stderr の ASCII から EBCDIC への変換は、 ILE 環境変数 QIBM_USE_DESCRIPTOR_STDIO を Y に、 QIBM_PASE_DESCRIPTOR_STDIO を B に設定すると、使用不可になります (バイナリー・データは stdin から読み取られ、 stdout または stderr に書き込まれます) 。 QIBM_PASE_DESCRIPTOR_STDIO のデフォルトは T (テキスト) です。この値を設定すると、EBCDIC から ASCII への変換が行われます。

ファイル・システムの詳細については、 統合ファイル・システムのトピックを参照してください。

グローバリゼーション

» OS/400 PASE ランタイムは AIX のランタイムをベースにしているため、OS/400 PASE プログラムでは、 AIX でサポートされている、ロケール、文字ストリング処理、日時サービス、メッセージ・カタログ、および文字エンコード変換といった、数多くの一連のプログラミング・インターフェースを使用することができます。 «

OS/400 PASE は、アプリケーションで使用するロケールの管理や、ロケールを区別する関数 (ctype や strcoll など) の実行において、 AIX ランタイムのインターフェースをサポートしています。これには、単一バイトとマルチバイト両方の文字エンコード方式のサポートも含まれます。

OS/400 PASE には AIX ロケールのサブセットが組み込まれており、これによって、業界標準のエンコード方式 (コード・セット ISO8859-x)、コード・セット IBM-1250、およびコード・セット UTF-8 を使用した、多くの国や言語のサポートが提供されます。 »OS/400 PASE は、 IBM-1252 ロケールと ISO 8859-15 ロケール (これらはいずれも単一バイトのエンコード方式を使用)、および UTF-8 ロケールという 3 つの異なる方法でユーロをサポートしています。 «

> **注:** OS/400 PASE のロケール・サポートは、 ILE C プログラムで使用されるロケール・サポート (オブジェクト・タイプ *CLD および *LOCALE) のいずれの形式にも属しません。内部構造が異なる上、ILE C プログラム用に同梱されている既存のロケールに、ASCII をサポートするものはありません。

新規ロケールの作成

OS/400 PASE には、新規ロケールを作成するためのユーティリティーは同梱されていません。ただし、localedef ユーティリティーを使用すれば、 AIX システム上の OS/400 PASE で使用するロケールを作成することは可能です。

ロケールの変更

OS/400 PASE アプリケーションでロケールを変更する場合は、一般的には、新しいロケールのエンコード方式と一致させるために、 OS/400 PASE CCSID も (_SETCCSID ランタイム機能を使用して) 変更するべきです。こうすることにより、すべての文字データ・インターフェース引き数が、 OS/400 PASE ランタイムによって正しく解釈されるようになります (また、場合によっては、EBCDIC システム・サービスの呼び出し時に変換されます)。 CCSID がどのコード・セット名と対応するかを確認するには、cstoccsid ランタイム機能を使用できます。

OS/400 PASE ランタイムは、OS/400 PASE プログラムによって作成されるすべてのファイルの CCSID タグを、現行の OS/400 PASE CCSID 値 (プログラムの開始時や最新の _SETCCSID 値を使用したときに得られる) に設定します。

日本語、韓国語、繁体字の中国語、および簡体字の中国語をサポートする OS/400 PASE には、UTF-8 ロケールを使用してください。 OS/400 には、これらの言語をサポートする他のロケールもありますが、システムは、OS/400 PASE CCSID を IBM-euc*XX* コード・セットと対応させる設定をサポートしていません。アプリケーションが他のプラットフォームで実行されるときは、ファイル・データが他のエンコード方式 (Shift-JIS など) で保管されている可能性もあるため、 UTF-8 サポートを使用するには、そのようなファイル・データの変換が必要になることがあります。

OS/400 PASE 変換オブジェクトとロケールの保管場所

»OS/400 PASE の変換オブジェクトとロケールは、 OS/400 言語フィーチャー・コードと一緒にパッケージされています。ロケールは、OS/400 PASE がインストールされるときに、インストールされる OS/400 言語フィーチャーに関連付けられているものだけが作成されます。 «

すべての OS/400 PASE ロケールでは、ASCII または UTF-8 の文字エンコード方式が使用されます。したがって、すべての OS/400 PASE ランタイムは、ASCII (または UTF-8) で動作します。

OS/400 のグローバリゼーションについては、 グローバリゼーションのトピックを参照してください。

メッセージ・サービス

OS/400 PASE のシグナルと ILE のシグナルは独立しているため、一方のタイプのシグナルを出してもう一方のシグナル・タイプのハンドラーを直接呼び出すことはできません。受け取った任意の ILE シグナルに対応する OS/400 PASE シグナルを POST するには、 OS/400 PASE Qp2SignalPase() API を使用できます。 QP2SHELL() プログラムと OS/400 PASE `fork()` 関数は、すべての ILE シグナルを対応する OS/400 PASE シグナルにマップするハンドラーを必ずセットアップします。

» システムは、Qp2RunPase、Qp2CallPase、または Qp2CallPase2 API を実行する呼び出しのプログラム・メッセージ・キューに送信されるすべての OS/400 例外メッセージを、自動的に、対応する OS/400 PASE シグナルに変換します。このようにして、OS/400 PASE アプリケーションは、システムによって変換された OS/400 PASE シグナルを処理することによって、すべての OS/400 例外を処理することができます。 «

» OS/400 PASE には、 OS/400 のメッセージ処理を直接管理できるようにする、次のようなランタイム機能があります。

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

これらの機能に関する詳細は、 ランタイム機能 (Run-time functions) を参照してください。 «

OS/400 メッセージ・サポート

OS/400 には、さまざまなコンテキストでのメッセージ・サポートがあります。

ジョブ・ログ
ジョブ・ログには、OS/400 やアプリケーションが実行されたりコンパイルされたりしたときに発行された、すべてのメッセージが記録されています。ジョブ・ログを表示するには、コマンド行から DSPJOBLOG と入力してください。「ジョブ・ログの表示 (Display Job Log)」画面が表示されたら、F10 を押し、続いて Shift + F6 を押します。これらのキーの組み合わせを押すと、「すべてのメッセージを表示 (Display All Messages)」画面が表示され、最新のメッセージに設定されます。特定のメッセージの詳細情報を表示するには、詳細を知りたいメッセージの上にカーソルを持っていき、F1 キーを押します。

アクティブ・ジョブの処理
OS/400 上のジョブやジョブ・スタックについて調べるには、 アクティブ・ジョブの処理 (WRKACTJOB) コマンドが役立ちます。

OS/400 でのメッセージ・サポートの詳細については、 実行管理機能のトピックを参照してください。

OS/400 PASE や OS/400 のメッセージの詳細については、 OS/400 PASE シグナル処理 (OS/400 PASE Signal Handling) を参照してください。

OS/400 PASE アプリケーションからの印刷出力

OS/400 PASE シェルからの出力の読み取りおよび書き込みを行うには、 QShell Rfile ユーティリティーを使用できます。

次の例では、ストリーム・ファイル mydoc.ps の内容を、スプールされているプリンター・ファイル QPRINT に未変換の ASCII データとして書き込み、 CL LPR コマンドを使用してそのスプール・ファイルを別のシステムに送信します。

```
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
```

疑似端末 (PTY)

OS/400 PASE は、AT&T スタイルと、バークレー・ソフトウェア・ディストリビューション (BSD) の、両方のスタイルのデバイスをサポートしています。プログラミングの観点からすれば、これらのデバイスは、AIX 上で機能するのと同じように OS/400 PASE 上で機能します。

OS/400 PASE では、AT&T スタイルのデバイスに最大 1024 のインスタンス、 BSD スタイルのデバイスに最大 592 のインスタンスを持つことができます。システムが開始されると、最初の 32 のインスタンスが、各デバイス・タイプに自動的に作成されます。

OS/400 PASE における PTY デバイスの構成

AIX において、管理者は、smit を使用して使用可能な各タイプのデバイスの数を構成します。一方 OS/400 PASE では、これらのデバイスは次のような方法で構成されます。

- AT&T スタイルのデバイスの場合、OS/400 PASE は自動構成をサポートしています。最初の 32 のインスタンスが使用されている状態でアプリケーションが別のインスタンスを開こうとすると、 1024 のデバイスを限度として、統合ファイル・システム内に自動的に CHRSF デバイスが作成されます。
- BSD スタイルのデバイスの場合は、OS/400 PASE mknod ユーティリティーを使用して、手動で CHRSF デバイスを作成する必要があります。これを行うためには、BSD スレーブ・デバイスと BSD マスター・デバイスの主要な番号と、命名規則を知っている必要があります。次の例は、追加の BSD PTY デバイスを作成する方法を示す、シェル・スクリプトです。この例では、グループ 16 にデバイスを作成します。

```
#!/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
```

疑似端末デバイスに関する詳細については、 AIX documentation  の Web サイトを参照してください。

セキュリティー

セキュリティーの視点から、 OS/400 PASE プログラムは、OS/400 上の他のすべてのプログラムと同じセキュリティー制限に属しています。 OS/400 で OS/400 PASE を実行するためには、統合ファイル・システム内の AIX バイナリーに対する権限が必要です。また、プログラムがアクセスするそれぞれのリソースに適したレベルの権限を持っていることも必要です。適切なレベルの権限がなければ、それらのリソースにアクセスしようとすると、プログラムはエラーを戻します。

OS/400 PASE プログラムを実行する場合には、次の情報が特に重要です。

ユーザー・プロファイルと権限管理

システムの権限管理は、オブジェクトの 1 つであるユーザー・プロファイルに基づいています。システム上で作成されるすべてのオブジェクトは、特定のユーザーによって所有されます。オブジェクトに対するそれぞれの操作やアクセスは、ユーザーの権限を確認するために、システムによって検証されます。所有者や適切な権限のあるユーザー・プロファイルは、他のユーザー・プロファイルに、オブジェクトを操作するさまざまなタイプの権限を委任することができます。権限検査は、すべてのタイプのオブジェクトに対して一様に行われます。

オブジェクトの権限のメカニズムによって、さまざまなレベルの制御が備えられます。ユーザーの権限は、必要なものだけに制限できます。 QOpenSys ファイル・システムに保管されるファイルには、UNIX ファイルと同じ方法で権限を付与できます。次の表は、UNIX の許可と、OS/400 データベース・ファイルで使用されるセキュリティー値の関係を示すものです。 OS/400 において、*OBJOPR はオブジェクトの*使用*権限を表し、*EXCLUDE は*権限なし* を表します。 *READ、*ADD、*UPD、*DLT、および *EXECUTE

はデータ権限です。ファイルを OS/400 PASE プログラムとして実行するには、そのファイルに対する *EXECUTE 権限 (および場合によっては *READ 権限) が必要です。

UNIX の許可	*OBJOPR	*READ	*ADD	*UPD	*DLT	*EXECUTE
r (読み取り)	X	X	-	-	-	-
w (書き込み)	X	-	X	X	X	-
x (実行)	X	-	-	-	-	X
権限なし	-	-	-	-	-	-

OS/400 PASE のユーザー・プロファイル

OS/400 では、認証情報は、/etc/passwd のようなファイルにではなく、個々の*プロファイル* に保管されます。ユーザーやグループにはプロファイルがあります。これらのプロファイルはすべて 1 つのネーム・スペースを共用しており、それぞれのプロファイルには、大文字のみを使用した固有な名前がなければなりません。 getpwnam() や getgrnam() といった API に小文字の名前が渡された場合、システムは名前のストリングを大文字に変換します。

» getpwuid() や getgrgid() を呼び出してプロファイル名を戻させるときは、 OS/400 PASE 環境変数が結果を大文字で戻す PASE_USRGRP_LOWERCASE=N に設定されていないと、結果が小文字になります。 «

すべてのユーザーには、ユーザー ID (UID) があります。また、すべてのグループには、グループ ID (GID) があります。これらは、POSIX 1003.1 規格に従って定義されます。この 2 つの数値スペースは分かれているので、ユーザーの UID を 104 として、グループの GID を 104 としても、それらの ID はそれぞれ区別されます。

OS/400 には、機密保護担当者 QSECOFR 用のユーザー・プロファイルがあります。このユーザーの UID は 0 です。他のどのプロファイルも UID を 0 にすることはできません。 QSECOFR は、システム上で最高の特権を持つプロファイルであり、その意味では root ユーザーとして動作します。ただし、OS/400 には、システム管理者によって個々のユーザーに割り当てることのできる、一連の特殊特権があります。このような特権の 1 つは *ALLOBJ で、この特権は、ファイル・アクセスに対する任意アクセス制御をオーバーライドします (たとえば、 UNIX システムでは、これを root 特権として使用するのが一般的です)。

root アクセスを使用する移植アプリケーションでは、 *ALLOBJ 権限を付与するアプリケーション・ユーザー 用に特定のユーザー・プロファイルを作成して、 QSECOFR を使用させない方が、セキュリティーの点では有利でしょう。 QSECOFR には、単一のアプリケーションで必要とされる以上の特権があるからです。 UNIX システムとは異なり、OS/400 では、ユーザーのグループ・メンバーシップは必要ありません。 OS/400 では、ユーザー・プロファイルの GID が 0 であることは、より多くの特権があるグループを表すのではなく、*グループの割り当てがない* ことを意味します。

OS/400 のセキュリティーは、システム内に構築される統合セキュリティーに依存しています。オブジェクトへのすべてのアクセスは、セキュリティー検査に通る必要があります。セキュリティー検査は、アクセス時にプロセスを実行するユーザー・プロファイルに関して実行されます。

OS/400 PASE は、保全性とセキュリティーの保守を、各プロセスに別個のアドレス・スペースを与えることに依存しています。 OS/400 PASE アドレス・スペースでリソースが使用できない場合は、そのリソースにはアクセスできません。ファイル・システムのセキュリティーは、適切な権限なしにユーザーがそのアドレス・スペースにリソースをロードするのを防ぎます。リソースは、一度アドレス・スペースに入ってしまうと、プロセスが実行される ID であるかどうかに関係なく処理できるようになります。

OS/400 PASE プログラムは、システム関数の要求にシステム呼び出しを使用します。 OS/400 PASE プログラムに対するシステム呼び出しは、OS/400 によって処理されます。このインターフェースでは、OS/400 PASE プログラムは、間接的 (かつ安全) な方法でしかシステム内部にアクセスできません。

iSeries サーバーのセキュリティーに関する詳細は、 セキュリティーのトピックを参照してください。

実行管理機能

OS/400 は、システム上の他のジョブを処理するのと同じ方法で OS/400 PASE プログラムを処理します。OS/400 がジョブを処理する方法の詳細については、 実行管理機能のトピックを参照してください。

OS/400 PASE のトラブルシューティング

以下の情報では、 OS/400 PASE プログラムをデバッグする方法と、それらをより効果的かつ有効に実行する方法について説明しています。

- OS/400 PASE プログラムのデバッグ
- パフォーマンスの最適化

OS/400 PASE プログラムのデバッグ

OS/400 PASE ランタイム環境は、`syslog()` ランタイム機能、および (より複雑なメッセージ・ルーティングのための) `syslogd` バイナリーのライブラリー・サポートを提供しています。加えて、OS/400 の既存の機能、たとえば診断メッセージ用のジョブ・ログや、 OS/400 システム・オペレーター・メッセージ・キュー QSYSOPR への重大メッセージの送信などを使用することができます。

アプリケーションによっては、 OS/400 PASE アプリケーションをデバッグするストラテジーで、異なるパスを使用することができます。

- アプリケーションが OS/400 の統合 (たとえば、 DB2 UDB for iSeries 関数または ILE 関数との統合) を必要としない場合、まず AIX 上でアプリケーションをデバッグする必要があります。
- その後、OS/400 PASE `dbx` と OS/400 デバッグ機能 (ジョブ・ログなど) を組み合わせて使用して、OS/400 上でアプリケーションをデバッグします。

データベースまたは ILE 関数を使用するようにコーディングしたアプリケーションを AIX 上で完全にテストすることはできませんが、 AIX 上でアプリケーションの残りの部分をデバッグすることにより、構造と設計が適切なものとなるようにすることができます。

OS/400 PASE での dbx の使用

OS/400 PASE は、AIX `dbx` デバッガー・ユーティリティーをサポートします。このユーティリティーを使用すると、親プロセスや子プロセスなどの関連プロセスを、ソース・コード・レベルでデバッグすることができます (そのようにコンパイルされた場合)。 OS/400 PASE で実行されるデバッガーに AIX ソースが見えるようにするために、ネットワーク・ファイル・システム (NFS) を使用することができます。

» xterm および aixterm 用の OS/400 PASE サポートにより、 `dbx` を使用して親プロセスと子プロセスの両方をデバッグすることができます。 `dbx` は、 `dbx` を 2 番目のプロセスに付加した別の `xterm` ウィンドウを立ち上げます。 «

`dbx` の詳細は、 AIX documentation の Web サイトを参照してください。 `dbx` コマンド行で `help` を入力することもできます。

OS/400 デバッグ・ツールの使用

ILE C ソース・デバッガーは、コードにおける問題を判別する上で効果的なツールです。このツールについては、 WebSphere(TM) Development Studio ILE C/C++ Programmer's Guide を参照してください。

OS/400 PASE コードをデバッグするには、 iSeries System Debugger も使用できます。

パフォーマンスの最適化

最適なパフォーマンスを得るため、アプリケーション・バイナリーは、ローカル・ストリーム・ファイル・システムに保管するようにしてください。バイナリー (基本プログラムとライブラリー) がローカル・ストリーム・ファイル・システムの外部にあると、ファイル・マッピングが行えないため、OS/400 PASE プログラムの開始がかなり遅くなります。

多くの `fork()` 操作を実行する OS/400 PASE でアプリケーションを実行すると、AIX で実行するときに比べて速度が落ちます。これは、それぞれの OS/400 PASE `fork()` 操作ごとに新しい OS/400 ジョブが開始されることで、パフォーマンスがかなりの影響を受ける可能性があるためです。

パフォーマンス・データの収集と分析については、『システム管理』のカテゴリーにあるパフォーマンスのトピックを参照してください。

例

以下の例が OS/400 PASE 情報として提供されています。これらの例を使用する前に、以下のコード例に関する特記事項をお読みください。

ILE プログラムからの OS/400 PASE プログラムおよびプロシージャーの実行

- ILE プログラムからの OS/400 PASE プログラムの実行
- ILE プログラムからの OS/400 PASE プロシージャーの呼び出し

OS/400 PASE プログラムからの OS/400 プログラムの呼び出し

- OS/400 PASE プログラムからの ILE プロシージャーの呼び出し
- OS/400 PASE からの OS/400 プログラムの呼び出し
- OS/400 PASE からの CL コマンドの実行

OS/400 PASE プログラムでの DB2 UDB for iSeries 関数の使用

- OS/400 PASE プログラムでの DB2 UDB for iSeries 呼び出しレベル・インターフェースの呼び出し

コードの特記事項情報

IBM は、お客様に、すべてのプログラミング・コード・サンプルを使用することができる非独占的な使用権を許諾します。お客様は、このサンプル・コードから、お客様独自の特別のニーズに合わせた類似のプログラムを作成することができます。

すべてのサンプル・コードは、例として示す目的でのみ、IBM により提供されます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。したがって IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

ここに含まれるすべてのプログラムは、現存するままの状態で提供され、いかなる保証条件も適用されません。第三者の権利の不侵害、商品性、特定目的適合性に関する黙示の保証の適用もいっさいありません。

OS/400 PASE の関連情報

OS/400 PASE の使用方法の詳細については、以下の資料を参照してください。

iSeries Information Center のその他のトピック

OS/400 PASE API

OS/400 PASE API の以下の一般カテゴリーの詳細については、このトピックを参照してください。

- 呼び出し可能プログラム API
- ILE プロシージャー API
- OS/400 PASE プログラムが使用するためのランタイム機能

OS/400 PASE プログラムを実行するには、システム API を呼び出す必要があります。システムでは、OS/400 PASE プログラムを実行するために、呼び出し可能プログラム API と ILE プロシージャー API の両方を提供しています。呼び出し可能プログラム API の使用は簡単ですが、 ILE プロシージャー API で使用可能なすべての制御を提供するわけではありません。

OS/400 PASE シェルおよびユーティリティー

OS/400 PASE には 3 つのシェル (Korn、Bourne、および C シェル) 、および OS/400 PASE プログラムとして実行する 200 個近くのユーティリティーがあります。 OS/400 PASE シェルおよびユーティリティーでは拡張可能なスクリプト環境を提供しており、その環境には業界標準の defacto-standard コマンドが多数含まれています。

OS/400 PASE コマンド

このトピックで説明されている OS/400 PASE コマンドのほとんどは、AIX コマンドと同じオプションをサポートしており、 AIX コマンドと同じ動作をします。 OS/400 PASE コマンドに加えて、各 OS/400 PASE シェルは多数の組み込みコマンド (`cd`、`exec`、`if` など) をサポートします。

OS/400 PASE ライブラリー (OS/400 PASE libraries)

OS/400 PASE ランタイムは、AIX ランタイムで提供される大規模なインターフェースのサブセットをサポートします。 OS/400 PASE でサポートされるランタイム・インターフェースのほとんどは、AIX と同じオプションおよび動作を提供します。 OS/400 PASE ランタイム・ライブラリーは `/usr/lib` に (シンボリック・リンクとして) インストールされます。

Web サイト

AIX コマンド、APIs、および OS/400 PASE がサポートするユーティリティー : API Analysis Tool は、OS/400 PASE によってサポートされている AIX アプリケーションの使用方法についての詳細情報を見つけるのに、効率的で有効です。

Developers Web サイトのIBM PartnerWorld における Application Factory および OS/400 PASE ページは、OS/400 PASE iSeries サーバーにアプリケーション移植する情報を提供します。 Application Factory は、 アプリケーションを iSeries サーバーに移植する他のソリューションと OS/400 PASE とを比較します。

AIX コマンドおよびユーティリティーの詳細については、AIX documentation  Web サイトを参照してください。

Printed in Japan

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12