IBM

IBM i
プログラミング
IBM PASE for i

7.1

IBM

IBM i
プログラミング
IBM PASE for i

7.1

ご注意

本書および本書で紹介する製品をご使用になる前に、 81 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM AIX 6 Technology Level 1 および IBM i 7.1 (製品番号 5770-SS1)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。このバージョンは、すべての RISC モデルで稼働するとは限りません。また CISC モデルでは稼働しません。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： IBM i
Programming
IBM PASE for i
7.1

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2010.4

目次

IBM PASE for i

IBM® Portable Application Solutions Environment for i (PASE for i) を使用すれば、最小限の労力で IBM AIX® アプリケーションを IBM i プラットフォームに移植できます。

AIX または Linux® などのオペレーティング・システムの管理の複雑さに煩わされることなく、選択したアプリケーションを実行できる統合されたランタイム環境が PASE for i によって提供されます。 PASE for i には、強力なスクリプト環境を提供する、業界標準やデファクト・スタンダードのシェルとユーティリティーも備えられています。

注: コード例を使用することにより、 79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

IBM i 7.1 の新機能

PASE for i のトピック集の中で、新規または大幅に変更された情報を以下に記載しています。

IBM i 7.1 の PASE for i で行われた変更は、次のとおりです。

- PASE for i は、AIX 6.1、Technology Level 2 から派生しています。
- PASE for i は、スタック実行使用不可保護を実施します。

システム・セキュリティーを改善するため、PASE for i プログラムの動作が変更され、プロセスのメモリー・エリア (スタック、ヒープ、および共有メモリー) からの命令実行をブロックできるようになりました。AIX では、この機能をスタック実行使用不可 (Stack Execution Disable: SED) サポートといいます。書き込み可能なメモリーにある命令の実行をブロックすることで、一般にインターネット・サーバーが標的になる、最も深刻なバッファー・オーバーフローのセキュリティー・アタックの多くに対抗できます (通常、そのためにサーバー・プログラムを変更する必要はありません)。

メモリー・エリアから取り出した命令を実行する必要がある PASE for i プログラムには、メインプログラムのヘッダーにビットを設定することでマークを付けなければなりません。プログラムにマークを付けて、メモリー・エリアからの命令の実行を許可するには、オブジェクト・ファイルのリンク時に `-brwexec_must` オプションを使用します。既存のバイナリーは、ldedit ユーティリティーを使用して変更できます。

```
ldedit -brwexec_must program_path_name
```

前のリリースの PASE for i は、SED サポート用のビットを無視します。このため、変更されたバイナリーは、それらのリリースでもそのまま実行できます。

`-brwexec_must` のマークが付けられていない PASE for i プログラムで、メモリー・エリアにある命令の実行が試行されると、ジョブ・ログに MCH6801 例外と理由コード 5 が生成され、PASE for i プログラムに SIGILL シグナルが通知されます。

メモリー・エリアにある命令を実行する必要がある一般的なアプリケーションの 1 つに IBM Technology for Java™ があります。IBM 提供の Java コマンドには、メモリー・エリアから命令を実行できるようにマークが付けられています。しかし、API を使用して既存の PASE for i 対応プロセス内で Java を起動するアプリケーションについては、必ずメインプログラムに適切にマークを付けて、メモリー・エリアからの命令の実行を許可する必要があります。

- AIX 6.1 ローダーに基づいた新規 PASE for i ローダー。

  新規ローダーは、LDR_CNTRL 環境変数の NAMEDSHLIB および HUGE_EXEC のオプションとスレッド固有ストレージの拡張サポートという方法を通じて、IBM POWER6™ または IBM POWER7 プロセッサー・ベースのサーバー上での 64 KB ページ・サイズ、64 ビット共用ライブラリーのキャパシティー拡大、および 32 ビットでのキャパシティー拡大をサポートします。
- /dev/urandom 文字特殊ファイルを疑似乱数のソースとして使用できます。
- PASE for i アプリケーションで使用できるように International Components for Unicode (ICU) C ライブラリーが提供されるようになりました。
- 表 1 に新規ユーティリティーをリストします。

表 1. 新規ユーティリティー

ldedit	XCOFF 実行可能ファイル・ヘッダーを変更します。
slibclean	カーネルおよびライブラリー・メモリー内の現在未使用のモジュールを除去します。

新規情報または変更情報の見分け方

技術上の変更が加えられた場所を見分けるのに役立つように、Information Center では以下のイメージを使用しています。

- » イメージにより、新規または変更された情報の開始点を示します。
- « イメージにより、新規または変更された情報の終了点を示します。

PDF ファイルでは、左マージンに新規および変更情報のリビジョン・バー (|) があります。

今回のリリースの新規情報または変更情報に関するその他の情報は、プログラム資料説明書を参照してください。

関連概念

12 ページの『IBM i での AIX コンパイラーのインストール』
これらの AIX コンパイラーを使用して、ご使用のシステムの PASE for i 環境内で、PASE for i アプリケーションの開発、コンパイル、ビルドおよび実行をすべて行うことができます。

63 ページの『IBM PASE for i シェルおよびユーティリティー』
PASE for i には、3 つのシェル (Korn 、Bourne、および C shell) および PASE for i プログラムとして実行する多数のユーティリティーがあります。PASE for i 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およびデファクト・スタンダードのコマンドが多数含まれています。

関連情報

IBM PASE for i プログラムが使用するためのランタイム機能

IBM PASE for i ロケール

IBM PASE for i の PDF ファイル

PASE for i の情報の PDF ファイルを表示または印刷できます。

本書の PDF 版を表示またはダウンロードするには、PASE for i を選択します。

PDF ファイルの保存

表示または印刷のために PDF をワークステーションに保存するには、以下のようにします。

1. ご使用のブラウザーで PDF リンクを右クリックする。
2. PDF をローカルに保存するオプションをクリックする。
3. PDF を保存したいディレクトリーに進む。
4. 「**保存**」をクリックする。

Adobe Reader のダウンロード

これらの PDF を表示または印刷するには、Adobe® Reader がシステムにインストールされていることが必要です。 このアプリケーションは、Adobe Web サイト (www.adobe.com/products/acrobat/readstep.html) から無料でダウンロードできます。

関連資料

78 ページの『IBM PASE for i の関連情報』

IBM Redbooks® 資料、Web サイト、およびその他の Information Center トピック集には、PASE for i のトピック集に関連した情報が揃っています。以下の PDF ファイルのいずれも表示または印刷できます。

IBM PASE for i の概要

PASE for i を使用すれば、わずかな変更で、またはまったく変更しなくても、IBM i オペレーティング・システム上の多数の AIX アプリケーション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。

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。同じように重要なこととして、IBM i サーバーの特長である機能が使いやすいことや統合しやすいことが挙げられます。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が、 AIX と IBM i 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。

これは、まさに PASE for i が得意とする分野です。

IBM PASE for i の概念

PASE for i は、IBM i オペレーティング・システム上で稼働する AIX アプリケーション用の統合ランタイム環境です。

PASE for i 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、AIX の共用ライブラリー、シェル、およびユーティリティーに備わったサポートの包括的なサブセットを提供します。 PASE for i はまた、IBM PowerPC® のマシン・インストラクションの直接処理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。

PASE for i アプリケーションには、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。
- IBM i 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 IBM i システム機能を使用します。

PASE for i は IBM i オペレーティング・システムでの 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。 PASE for i は、ほとんどあるいはまったく変更しなくても、 IBM i オペレーティング・システム上で AIX プログラムを実行するように設計されています。 UNIX または Linux などの他の環境からのプログラムは、 PASE for i 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。

PASE for i 統合ランタイムは、IBM i オペレーティング・システム上のライセンス内部コード カーネルで実行します。 システムは、IBM i およびその他の実行時環境 (Integrated Language Environment® (ILE) や Java も含む) にまたがって、共通した多数の PASE for i 機能を統合します。 PASE for i 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。 PASE for i のシステム・サポートでは、PASE for i プログラムがどのメモリーにアクセスできるかを制御し、特権付きでないマシン・インストラクションのみの使用に制限することにより、システム・セキュリティーと整合性が配備されます。

最小限の労力での迅速なアプリケーション展開

多くの場合、AIX プログラムはほとんどあるいはまったく変更せずに、 PASE for i 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) IBM i 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。

PASE for i では、 IBM i 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。 PASE for i 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、市場参入までの期間が改善され、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。

IBM i での AIX テクノロジーの幅広いサブセット

PASE for i 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、PASE for i 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てから、IBM i オペレーティング・システムで実行されます。

もう一つの方法として、 PASE for i 環境でサポートされているコンパイラー製品の 1 つをインストールして、 PASE for i 内で完全にアプリケーションを開発、コンパイル、ビルド、および実行することもできます。

PASE for i 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。

PASE for i は、AIX および IBM i 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、 IBM i モードから AIX モードへ切り替えて、 PASE for i ランタイムでアプリケーションを実行します。

PASE for i で実行するアプリケーションは、 IBM i 統合ファイル・システムおよび DB2® for i に統合されます。それらは Java および 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。一般的にそれらは、セキュリティー、メッセージ処理、通信、およびバックアップとリカバリーなどの、IBM i オペレーティング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。

関連概念

63 ページの『IBM PASE for i シェルおよびユーティリティー』
PASE for i には、3 つのシェル (Korn 、Bourne、および C shell) および PASE for i プログラムとして実行する多数のユーティリティーがあります。PASE for i 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およびデファクト・スタンダードのコマンドが多数含まれています。

関連資料

10 ページの『AIX ソースのコンパイル』
IBM i でのインストールをサポートする AIX コンパイラー製品のいずれかをインストールして、PASE for i 環境でプログラムをコンパイルすることができます。

アプリケーション開発における有望な選択肢としての IBM PASE for i

API 分析ツールを使用して、アプリケーションが PASE for i に適しているかどうかを判別することができます。PASE for i は、一部の環境では最良のソリューションとはなりません。

AIX アプリケーションをシステムに移植する方法について、PASE for i を介して非常に柔軟に決定を下すことができます。当然、PASE for i は、AIX アプリケーションの移植で使用できるいくつかのオプションのうちの 1 つにすぎません。

API の分析

アプリケーションが PASE for i 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 IBM i オペレーティング・システムでどれほど効率的に実行されるかを分析します。アプリケーションを PASE for i に移植する際の手順に、コード分析をどのように組み込むかについては、 9 ページの『IBM PASE for i で実行するプログラムの準備』を参照してください。

PASE for i 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、PASE for i 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。

- **その AIX アプリケーションは主に計算を軸としたアプリケーションであるか?**

  PASE for i 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、 IBM i オペレーティング・システム上で計算主体のアプリケーションを実行するために適した環境を提供します。
- **そのアプリケーションは PASE for i だけで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) サポートなど) にかなり依存しているか?**

PASE for i は fork() および exec() 関数のサポートを提供しています。これらは、現在の IBM i オペレーティング・システムにはありません。(ただし、fork() 関数と exec() 関数を組み合わせる spawn() 関数を使用した場合は例外です。)

- **そのアプリケーションは複雑な AIX システム・ベースのビルド・プロセスまたはテスト環境を使用しているか?**

  PASE for i では AIX システム・ベースのビルド・プロセスが使えます。これは特に、新しいオペレーティング・システムに簡単には移せない、既存の複雑なプロセスがある場合に役立ちます。
- **そのアプリケーションは ASCII 文字セットに依存しているか?**

  PASE for i 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。
- **そのアプリケーションは多くのポインター操作を行うか、または整数からポインターへの変換 (キャスト) を行うか?**

  PASE for i 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。

PASE for i 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 PASE for i は適していません。

- 呼び出しごとに PASE for i を開始または終了するか、またはすでにアクティブな PASE for i プログラムで PASE for i 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。 PASE for i 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、PASE for i から呼び出される ILE コードでは、PASE for i メモリーを共用または使用できます。)

IBM PASE for i のインストール

PASE for i は、オプションでインストールできるオペレーティング・システム・コンポーネントです。これを使用したり、PASE for i サポートを必要とするソフトウェアを実行するには、PASE for i をインストールする必要があります。

拡張ドメイン・ネーム・システム (DNS) サーバーや 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、PASE for i サポートが必要です。したがって、PASE for i を直接使用する予定がなくても、やはり PASE for i をインストールする必要はあります。

PASE for i は、どの IBM i サーバーでも無償で利用できます。

PASE for i をシステムにインストールするには、次のステップを行います。

1. IBM i コマンド行で、`GO LICPGM` と入力します。
2. 11 (ライセンス・プログラムのインストール) を選択します。
3. オプション 33 (5770-SS1 - 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。
4. オプション: 追加ロケールをインストールします。

PASE for i 製品では、 IBM i オペレーティング・システム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。 システム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 IBM i 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です。

PASE for i 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::

PASE for i には、IBM i 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。 IBM i に同梱されているライブラリー・コードはすべて、IBM i のライセンスで使用できます。ただし、このライセンスは、 PASE for i 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 PASE for i に移植しようとするとき、そのアプリケーションが、 PASE for i 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 IBM i 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 IBM i 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。

関連概念

56 ページの『グローバリゼーション』

PASE for i は AIX のランタイムをベースにしているため、 PASE for i 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。

関連情報

IBM PASE for i ロケール

IBM PASE for i の計画

PASE for i は、最小限の手間で AIX アプリケーションをシステムに移植できるように IBM i オペレーティング・システム上に AIX 実行時環境を整備します。

実際、多くの AIX プログラムは、変更を加えなくても PASE for i で稼働します。これは、PASE for i 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、 System p® PowerPC プロセッサーで稼働するのと同じように直接 System i® 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。

PASE for i での作業時には、以下の点に留意してください。

- **ターゲットの AIX バイナリー・リリースと、バイナリーの稼働場所である PASE for i のリリースとの間には、相互関係があります。**

  PASE for i アプリケーションを AIX 上でコンパイルする場合、 AIX で作成するバイナリー・バージョンのアプリケーションには、そのアプリケーションを実行する PASE for i のバージョンとの互換性が必要です。次の表は、PASE for i 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 5.3 用に作成されたアプリケーションは、IBM i 5.4 およびそれ以降のリリースの PASE for i でのみ実行できます。同様に、AIX リリース 6.1 用に作成されたアプリケ

ーションは、IBM i 7.1 では実行できますが、それより前のリリースでは実行できません。

AIX のリリース	i5/OS® V5R3	IBM i V5R4	IBM i 6.1	IBM i 7.1
5.1 (32 または 64 ビット)	X	X	X	X
5.2 (32 または 64 ビット)	X	X	X	X
5.3 (32 または 64 ビット)		X	X	X
6.1 (32 または 64 ビット)				X

- **PASE for i では、IBM i オペレーティング・システム上に 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、IBM i カーネルか統合 IBM i 機能にルーティングされます。この点に関しては、PASE for i によって、AIX と IBM i オペレーティング・システムとの間のギャップが埋められます。共用ライブラリー内の API に関しては、AIX のものと同じ構文がコードで使用されますが、PASE for i プログラムは、 IBM i ジョブの中で実行され、他のすべての IBM i ジョブとまったく同じように IBM i オペレーティング・システムによって管理されます。
- **ほとんどのケースにおいて、PASE for i 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、PASE for i では違う動作をするか、あるいは PASE for i ではサポートされていません。このため、PASE for i プログラムの準備の計画を立てる際は、コード全体の分析をまず行う必要があります。コード分析の詳細については、 9 ページの『IBM PASE for i とのプログラムの互換性の分析』を参照してください。
- **AIX プラットフォームと IBM i 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/小文字の区別がありますが、特定の IBM i 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、IBM i オペレーティング・システムでは、通常は拡張 2 進化 10 進交換コード (EBCDIC) が使用されます。 PASE for i プログラムからの ILE コード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、PASE for i 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、ストリングに対して文字エンコード変換を処理するように、明示的に PASE for i プログラムをコーディングする必要があります。 PASE for i 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()、iconv_close()、および iconv_open() 関数も組み入れられています。

    注: iconv() インターフェースのインプリメントは、PASE for i 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。 PASE for i iconv() 関数でサポートされる変換は、統合ファイル・システム内にバイト・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) ソフトウェアおよび IBM i 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、 PASE for i で使用するパスを反映して、パスに変更を加えることが必要になる場合があります。 IBM i オペレーティング・システムとのプログラムの互換性を分析する必要があります。

これらの問題のいくつかは、PASE for i によって自動的に処理されます。たとえば、通常はファイル記述子 (バイト・ストリーム・ファイルまたはソケット) に読み書きされるデータに対しては何の変換も実行さ

れませんが、システムによって提供される PASE for i ランタイム・サービス (IBM i 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、PASE for i は、必要に応じて ASCII から EBCDIC への変換を実行します。

ILE 関数や API への呼び出しを行う PASE for i 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、データ変換を処理する必要がある場合があります。また、プログラムにこれらの拡張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。

関連概念

『IBM PASE for i とのプログラムの互換性の分析』
IBM i オペレーティング・システムへの C アプリケーションの移植性を評価する最初のステップとしては、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します。

IBM PASE for i で実行するプログラムの準備

IBM i オペレーティング・システムで効率よく稼働する AIX プログラムの準備のためにとるステップは、プログラムの特性や、IBM i システム固有のインターフェースや機能の実際の使用目的に応じて異なります。

PASE for i に 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でき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プログラムを修正する必要があります。

IBM PASE for i とのプログラムの互換性の分析

IBM i オペレーティング・システムへの C アプリケーションの移植性を評価する最初のステップとしては、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します。

- 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

  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確認

  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 PASE for i 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、 PASE for i は `/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 PASE for i は `/QOpenSys` ファイル・システムを探します。

- サポートされないシステム呼び出しの確認

  PASE for i カーネルは、AIX カーネルでインプリメントされていても PASE for i ではサポートされない一部のシステム呼び出しをエクスポートします。サポートされないシステム呼び出しに対するデフォ

ルトの動作は、サポートされないシステム呼び出しの名前が入った例外メッセージ MCH3204 を送信することです。さらに、システムによって PASE for i シグナル SIGILL が呼び出しプロセスに送信されます。

サポートされないすべてのシステム呼び出しの使用を避けるため、アプリケーション・ソースの変更が必要になる場合があります。PASE_SYSCALL_NOSIGILL 環境変数を使用して、サポートされないシステム呼び出しに対するシステムの動作を変更することもできます。

関連概念

7 ページの『IBM PASE for i の計画』
PASE for i は、最小限の手間で AIX アプリケーションをシステムに移植できるように IBM i オペレーティング・システム上に AIX 実行時環境を整備します。

関連情報

PASE for i 環境変数

AIX ソースのコンパイル

IBM i でのインストールをサポートする AIX コンパイラー製品のいずれかをインストールして、PASE for i 環境でプログラムをコンパイルすることができます。

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、 PASE for i 用のバイナリー・ファイルを作成します。 PASE for i 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。

PASE for i 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。

PASE for i (オペレーティング・システムのオプション 33) にはコンパイラーが含まれません。 AIX システムを使用して PASE for i プログラムをコンパイルするか、または PASE for i でのインストールをサポートする AIX コンパイラー製品のうちの一つをオプションでインストールしてPASE for i 環境でプログラムをコンパイルすることができます。

System p プラットフォームでの AIX コンパイラーの使用

PowerPC 用の AIX アプリケーション・バイナリー・インターフェース (ABI) と互換性がある出力を生成する任意の AIX コンパイラーおよびリンカーを使用して、PASE for i プログラムを作成することができます。 PASE for i は、PowerPC には存在しない POWER® アーキテクチャー指示 (キャッシュ管理の IBM POWER 指示は存在する) を使用するバイナリー・ファイルの指示エミュレーション・サポートを提供します。

PASE for i での AIX コンパイラーの使用

IBM i は、PASE for i 環境での以下の別々に使用可能な AIX コンパイラーのインストールをサポートします。

- IBM XL C/C++ for AIX
- IBM XL C for AIX
- IBM XL Fortran for AIX

これらの製品を使用して、ご使用のシステムの PASE for i 環境内で、PASE for i アプリケーションの開発、コンパイル、ビルドおよび実行のすべてを行うことができます。

開発ツール

PASE for i 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。PASE for i 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。

IBM Tools for Developers for i5/OS PRPQ (5799-PTL) にも、IBM i 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揃えられています。この PRPQ についての詳細は、IBM Tools for Developers for i5/OS の Web サイト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、マシン・インターフェース (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 PASE for i 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すべてのコマンドでデフォルトであるわけではありません)。

例

以下の例は、AIX システムで PASE for i 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。PASE for i にインストールされたコンパイラーを使用してプログラムのコンパイルを行っているのであれば、IBM i システム固有のヘッダー・ファイルまたは IBM i システム固有のエクスポート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。これらのファイルは IBM i システム上のデフォルト・パス位置 /usr/include/ および /usr/lib/ にあるからです。

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 PASE for i 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 IBM i システム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、IBM i システム固有のヘッダー・ファイルが AIX ディレクトリー /usr/include にコピーされ、 IBM i システム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、IBM i システム固有のヘッダー・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、IBM DB2 for i コール・レベル・インターフェース (CLI) 用の PASE for i 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。

関連概念

3 ページの『IBM PASE for i の概念』
PASE for i は、IBM i オペレーティング・システム上で稼働する AIX アプリケーション用の統合ランタイム環境です。

63 ページの『IBM PASE for i シェルおよびユーティリティー』
PASE for i には、3 つのシェル (Korn 、Bourne、および C shell) および PASE for i プログラムとして実行する多数のユーティリティーがあります。PASE for i 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およびデファクト・スタンダードのコマンドが多数含まれています。

関連情報

IBM Tools for Developers for IBM i の Web サイト

IBM i での AIX コンパイラーのインストール

これらの AIX コンパイラーを使用して、ご使用のシステムの PASE for i 環境内で、PASE for i アプリケーションの開発、コンパイル、ビルドおよび実行をすべて行うことができます。

次の別個に入手可能な AIX コンパイラーのいずれかを IBM i にインストールして、PASE for i 環境で使用できます。

- IBM XL C/C++ for AIX
- IBM XL C for AIX
- IBM XL Fortran for AIX

関連概念

1 ページの『IBM i 7.1 の新機能』
PASE for i のトピック集の中で、新規または大幅に変更された情報を以下に記載しています。

関連情報

XL C/C++ for AIX

XL C for AIX

インストール・メディアからの AIX コンパイラーのインストール:

PASE for i は、AIX を実行しているシステムにアプリケーションをインストールするために一般的に使用される、AIX の `smit` または `installp` ユーティリティーをサポートしていません。 XL C/C++ for AIX、XL C for AIX、または XL Fortran for AIX 製品のインストールは、各コンパイラーのインストール・メディアにある非デフォルト・インストール・スクリプトを使用して行います。

コンパイラーを正常にインストールして使用するには、コンパイラー・インストール・メディアに加えて、ご使用のシステムに以下のプログラムがインストールされている必要があります。

- 5770-SS1 オプション 33 - PASE for i
- 5770-SS1 オプション 13 - システム・オープンネス・インクルード。`/usr/include` 統合ファイル・システム・ディレクトリーにあるコンパイラー・ヘッダー・ファイルを含む。
- Perl

  コンパイラー・インストール・スクリプトには Perl が必要です。 Perl のインストールには、以下の 2 つの方法があります。

  – 5799-PTL - Tools for Developers for i5/OS PRPQ。これには Perl とその他多くの便利な開発ツールが組み込まれています。Tools for Developers for i5/OS PRPQ の追加情報については、IBM ポーティング・セントラル Web サイト (英語) (www.ibm.com/servers/enable/site/porting/tools/) を参照してください。

  – CPAN Perl Port (バイナリー配布) Web サイト (英語) (www.cpan.org/ports/#os400) - PASE for i 用 Perl Port のバイナリー配布。

IBM i に XL C/C++ for AIX、XL C for AIX、または XL Fortran for AIX 製品をインストールするには、次の手順に従ってください。

1. コンパイラー製品インストール CD を CD ドライブに挿入します。
2. システムに *ALLOBJ 権限のあるユーザー・プロファイルでサインオンします。 コンパイラー製品ファイルは、このユーザー・プロファイルに所有されます。
3. CL コマンド `call qp2term` を入力して、対話式 PASE for i 端末セッションを開始します。
4. 以下のコマンドを入力して、適当なコンパイラー・インストール・スクリプトを復元します。

コンパイラー	コマンド
XL C/C++ for AIX 用	`cd /` `restore -qf /QOPT/CDROM/USR/SYS/INST.IMA/VACPP.NDI ./usr/vacpp/bin/vacppndi`
XL C for AIX 用	`cd /` `restore -qf /QOPT/CDROM/USR/SYS/INST.IMA/VAC.NDI ./usr/vac/bin/vacndi`
XL Fortran for AIX 用	`cd /` `restore -qf /QOPT/CDROM/USR/SYS/INST.IMA/XLF.NDI ./usr/lpp/xlf/bin/xlfndi`

5. インストール・スクリプトを実行して、コンパイラーをインストールします。 コンパイラーの宛先ディレクトリーは、コマンドの -b オプションで指定します。次の表のコマンドでは、コンパイラー用に推奨されるディレクトリー名を使用しています。異なるディレクトリーを選択する場合、そのディレクトリーは (大/小文字の区別があるファイル名を使用できるように) /QOpenSys ツリー内になければなりません。

コンパイラー	コマンド
XL C/C++ for AIX 用	/QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vacpp/bin/vacppndi -i -d /QOPT/CDROM/USR/SYS/INST.IMA -b /QOpenSys/xlcpp
XL C for AIX 用	/QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vac/bin/vacndi -i -d /QOPT/CDROM/USR/SYS/INST.IMA -b /QOpenSys/xlc
XL Fortran for AIX 用	/QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/lpp/xlf/bin/xlfndi -i -d /QOPT/CDROM/USR/SYS/INST.IMA -b /QOpenSys/xlf
注: コマンドは 1 つの長いコマンドとして入力してください。	

これでコンパイラーはインストールされ、PASE for i で使用できるようになりました。

xlC などの XL C/C++ for AIX コンパイラー・コマンド (たとえば、xlc) は、ディレクトリー /QOpenSys/xlcpp/usr/vacpp/bin/ にあります。

XL C/C++ for AIX コンパイラーの資料 (PDF 形式) は、ディレクトリー /QOpenSys/xlcpp/usr/vacpp/doc/en_US/pdf にあります。

xlc や cc などの XL C for AIX コンパイラー・コマンドは、ディレクトリー /QOpenSys/xlc/usr/vac/bin/ にあります。

XL C for AIX コンパイラーの資料 (PDF 形式) は、ディレクトリー /QOpenSys/xlc/usr/vac/doc/en_US/pdf にあります。

XL Fortran for AIX コンパイラー・コマンド (たとえば、xlf) は、ディレクトリー /QOpenSys/xlf/usr/bin/ にあります。

XL Fortran for AIX コンパイラーの資料 (PDF 形式) は、ディレクトリー /QOpenSys/xlf/usr/lpp/xlf/doc/en_US/pdf にあります。

注: コンパイラー・コマンドが入っているディレクトリーは、$PATH 環境変数に追加できます。$PATH 環境変数は、コマンドを探すときに検索すべきディレクトリーを指定します。たとえば、XL C/C++ for AIX をインストールした場合は、次のように $PATH 環境変数を変更すると、コンパイラー・コマンドを使用するときにコマンド・パスの指定を回避できます。

```
export $PATH=$PATH:/QOpenSys/xlcpp/usr/vacpp/bin
```

PTF 更新手順:

XL C/C++ for AIX、XL C for AIX、または XL Fortran for AIX 製品のためのプログラム一時修正 (PTF) のインストールは、最初のコンパイラー・インストールで使用したものと同じ非デフォルト・インストール・スクリプトを使用して実行します。

PTF をインストールする前に、このトピック内の前述のステップを使用してコンパイラーのインストールを実行しておく必要があります。

IBM i に XL C/C++ for AIX、XL C for AIX、または XL Fortran for AIX 製品の PTF をインストールするには、次の手順に従ってください。

1. インストールする PTF パッケージ・ファイルを入手します。 コンパイラーの PTF パッケージの圧縮 TAR イメージは、製品 Web サイトのサポート・ダウンロード・セクションからダウンロードできます。
2. PTF パッケージ・ファイルを解凍し、次いで untar します。 圧縮 TAR イメージを /QOpenSys/vacptf/ ディレクトリーにダウンロードしたら、 QP2TERM コマンド行から以下のコマンドを使用してこれを実行できます。

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

3. インストールする PTF パッケージのリストを含むファイルを作成します。 そのためには、QP2TERM コマンド行で次のようなコマンドを使用します。

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

4. 必要な場合、非デフォルト・インストール (NDI) ツール自体を確認し、更新します。 NDI ツールの更新版を使用して、PTF パッケージの残りの部分をインストールするには、まずツールの更新版を復元する必要があります。下の表から、更新するコンパイラーに適用されるコマンドを使用してください。表の手順に従う際、次の点に注意してください。
   - 表中の ls コマンドで、ファイルが存在しないことを示すエラー・メッセージが返された場合、NDI ツールは PTF パッケージによって更新されていません。表の残りのコマンドはスキップして、これらの手順の次のステップを続行してください。
   - 表中の ls コマンドでファイル名が返された場合は、このファイル名をメモして、下の表の restore コマンドでそのファイル名を使用してください。たとえば、返されたファイル名が vacpp.ndi.09.00.0000.0006.bff だったとします。このとき、restore コマンドに表示されている vacpp.ndi.WW.XX.YYYY.ZZZZ.bff 部分を、返された実際の名前に置き換えると、restore コマンドは次のようになります。

     ```
     restore -qf vacpp.ndi.09.00.0000.0006.bff ./usr/vacpp/bin/vacppndi
     ```

コンパイラー	コマンド
XL C/C++ for AIX 用	cd / ls /QOpenSys/ptf/vacpp.ndi.* restore -qf *vacpp.ndi.WW.XX.YYYY.ZZZZ.bff* ./usr/vacpp/bin/vacppndi
XL C for AIX 用	cd / ls /QOpenSys/ptf/vac.ndi.* restore -qf *vac.ndi.WW.XX.YYYY.ZZZZ.bff* ./usr/vac/bin/vacndi
XL Fortran for AIX 用	cd / ls /QOpenSys/ptf/xlf.ndi.* restore -qf *xlf.ndi.WW.XX.YYYY.ZZZZ.bff* ./usr/lpp/xlf/bin/xlfndi

5. インストール・スクリプトを実行して、PTF をインストールします。更新するコンパイラーに応じて、以下のコマンドの 1 つを QP2TERM コマンド行から入力します。

コンパイラー	コマンド
XL C/C++ for AIX 用	/QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vacpp/bin/vacppndi -d /QOpenSys/ptf -b /QOpenSys/xlcpp -u /QOpenSys/ptf/ptflist.txt
XL C for AIX 用	/QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/vac/bin/vacndi -d /QOpenSys/ptf -b /QOpenSys/xlc -u /QOpenSys/ptf/ptflist.txt

コンパイラー	コマンド
XL Fortran for AIX 用	/QIBM/ProdData/DeveloperTools/pase/bin/perl /usr/lpp/xlf/bin/xlfndi -d /QOpenSys/ptf -b /QOpenSys/xlf -u /QOpenSys/ptf/ptflist.txt
注: コマンドは 1 つの長いコマンドとして入力してください。	

インストール・スクリプトは、PTF 更新前に存在していたコンパイラー・ファイルの圧縮 TAR バックアップを作成します。この説明で示されるディレクトリーを使用する場合、このファイル名は /QOpensys/xlcpp.backup.tar.Z、/QOpenSys/xlc.backup.tar.Z、または /QOpensys/xlf.backup.tar.Z となります。 PTF 更新のインストールまたは PTF 更新そのもので問題が発生した場合、このバックアップから復元して、PTF 更新をアンインストールすることができます。

関連情報

XL C/C++ for AIX

システムへの IBM PASE for i プログラムのコピー

PASE for i で実行する予定の AIX バイナリー・ファイルは、統合ファイル・システムにコピーする必要があります。

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、 PASE for i 内で使用することができます。

オペレーティング・システム間でファイルを移動するときは、アプリケーションでの大/小文字の区別、および AIX が使用する改行文字と IBM i オペレーティング・システムが使用する改行文字の違いを理解しておく必要があります。 これらの違いによって問題が発生する場合があります。

ファイル転送プロトコル (FTP)、Server Message Block (SMB)、またはリモート・ファイル・システムを使用することにより、PASE for i とシステムの間でプログラムや関連ファイルをやり取りすることができます。

関連資料

20 ページの『ヘッダー・ファイルのコピー』
この情報を参考にして、IBM i を実行しているシステムから AIX を実行しているシステムにヘッダー・ファイルをコピーします。

21 ページの『エクスポート・ファイルのコピー』
この情報を参考にして、IBM i を実行しているシステムから 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。

関連情報

統合ファイル・システム

大/小文字の区別

アプリケーションが大/小文字の区別を行う場合、/QOpenSys ファイル・システム、または大/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 および Linux などのオペレーティング・システムのインターフェースは一般に、大/小文字を区別します。 IBM i オペレーティング・システムでは、大/小文字の区別がない場合もあります。 特に、大/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/小文字の区別は、 IBM i オペレーティング・システム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/小文字の区別があり、大/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。

例

以下は、大/小文字の区別から生じる可能性のある問題の例です。

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/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 PASE for i 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
```

関連情報

ファイル・システムの比較

統合ファイル・システムにおける改行文字

AIX および IBM i オペレーティング・システムでは、たとえばファイルおよびシェル・スクリプト内などのように、テキスト・ファイル内で使用する改行文字が異なります。

PASE for i 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 IBM i 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、ファイル転送プロトコル (FTP) を使ってソース・ファイルとシェル・スクリプトを AIX オペレーティング・システムから IBM i オペレーティング・システムに転送す

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復帰が除去されます。 IBM i 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 PASE for i 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、AIX オペレーティング・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のファイルから CR を除去するための手段として、以下のコマンドを使用することができます。

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

IBM i や PC のエディターでの CRLF の使用

また、ご使用のシステムのエディターやワークステーションのエディター (Windows® のメモ帳エディターなど) 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、PASE for i 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)。

ファイルの転送

ファイル転送プロトコル (FTP)、Server Message Block (SMB)、またはリモート・ファイル・システムを使用することにより、PASE for i プログラムとその関連ファイルをご使用のシステムでやりとりすることができます。

- ファイル転送プロトコルを使用したプログラムのコピー
- Server Message Block を使用したプログラムのコピー
- 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

ファイル転送プロトコルを使用したプログラムのコピー

IBM i ファイル転送プロトコル (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、IBM i 統合ファイル・システム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 (IBM i 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、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。

- 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。

Server Message Block を使用したプログラムのコピー

IBM i オペレーティング・システムは、Server Message Block (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、 PASE for i 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。 AIX プラットフォームまたは Linux オペレーティング・システ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。 構成済みで操作可能なオペレーティング・システム (AIX など) をインストールすると、PASE for i 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

IBM i オペレーティング・システムでは、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™) と Andrew File System (AFS®) 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)、 IBM i オペレーティング・システム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、PASE for i アプリケーション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。 IBM i 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。プラットフォームが複数ある場合に、そのいずれでも同一のユーザーとみなされる必要のあるユーザー・プロファイルには、どのシステムでも必ず同じユーザー ID が付いているようにします。

IBM i オペレーティング・システムは 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。その場合、AIX オペレーティング・システムから IBM i 統合ファイル・システムのディレクトリーに NFS ファイル・システムをマウントする必要があります。すると、AIX は、プログラムが作成されたときにそのプログラムを IBM i オペレーティング・システムに直接書き込みます。

注: IBM i 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。

関連情報

ファイル転送プロトコル

IBM i 機能を使用するための IBM PASE for i 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 IBM i 共用ライブラリーでは直接サポートされていない PASE for i 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。

準備を行うには、以下のステップを完了してください。

1. PASE for i システム固有の機能へのアクセスを調整する、すべての必要な IBM i 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コーディングします。
2. AIX システム上の PASE for i 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。
   a. AIX システムに IBM i システム固有の必須ヘッダー・ファイルをコピーします。
   b. AIX システムに IBM i システム固有の必須エクスポート・ファイルをコピーします。

関連概念

34 ページの『IBM PASE for i プログラムからの IBM i プログラムおよびプロシージャーの呼び出し』
PASE for i では、ILE プロシージャー、Java プログラム、OPM プログラム、IBM i API、および IBM i 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。

45 ページの『IBM PASE for i プログラムと IBM i の相互作用』
PASE for i の機能を使用するように IBM i 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。

関連情報

IBM PASE for i プログラムが使用するためのランタイム機能

ヘッダー・ファイルのコピー

この情報を参考にして、IBM i を実行しているシステムから AIX を実行しているシステムにヘッダー・ファイルをコピーします。

PASE for i は、標準 AIX ランタイムに、IBM i システム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、PASE for i および IBM i オペレーティング・システムによって提供されます。

IBM i から、ヘッダー・ファイルの検索パス内の AIX へのヘッダー・ファイルのコピー

ヘッダー・ファイルは、AIX ディレクトリー /usr/include、またはコンパイラーのヘッダー・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 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。

PASE for i ヘッダー・ファイルのコピー

PASE for i ヘッダー・ファイルは、以下の IBM i ディレクトリーにあります: /QOpenSys/QIBM/ProdData/OS400/PASE/include。

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

ヘッダー・ファイル	説明
as400_protos.h	このヘッダー・ファイルは、PASE for i システム固有の各種機能を ILE に提供します。
as400_types.h	このヘッダー・ファイルは、ILE への呼び出し用の固有の IBM i パラメーター・タイプを宣言します。 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	このヘッダー・ファイルは、IBM i メッセージを送受信するための関数を宣言します。

IBM i ヘッダー・ファイルのコピー

IBM i アプリケーションで他の PASE for i 機能にアクセスする計画であれば、使用する IBM i 機能のためのヘッダー・ファイルを、開発マシンにコピーしておくと便利な場合があります。一般的には、IBM i プログラムまたはプロシージャーは、 PASE for i アプリケーションから直接実行することはできないことに注意してください。

IBM i システムで提供されるヘッダー・ファイルは、/QIBM/include ディレクトリーにあります。

アプリケーションで何らかの IBM i 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、 PASE for i の `Rfile` ユーティリティーを使用できます。

次に示す例では、PASE for i の `Rfile` ユーティリティーで IBM i ヘッダー・ファイル `/QIBM/include/qusec.h` を読み取り、そのデータを PASE for i コード化文字セット ID (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
```

関連概念

46 ページの『データベース』
PASE for i は、DB2 for i コール・レベル・インターフェース (CLI) をサポートします。 AIX および IBM i 上の DB2 CLI は、それぞれ互いのまったく同じ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ます。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。

34 ページの『IBM PASE for i プログラムからの IBM i プログラムおよびプロシージャーの呼び出し』
PASE for i では、ILE プロシージャー、Java プログラム、OPM プログラム、IBM i API、および IBM i 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。

関連タスク

34 ページの『ILE プロシージャーの呼び出し』
以下のステップを行って、ILE プロシージャーを準備し、PASE for i プログラムから呼び出すことができます。

関連資料

16 ページの『システムへの IBM PASE for i プログラムのコピー』
PASE for i で実行する予定の AIX バイナリー・ファイルは、統合ファイル・システムにコピーする必要があります。

エクスポート・ファイルのコピー

この情報を参考にして、IBM i を実行しているシステムから 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。

IBM i システム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 IBM i 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任意の 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。

PASE for i は以下のエクスポート・ファイルを提供します。

エクスポート・ファイル	機能
as400_libc.exp	このファイルは、libc.a にある、IBM i システム固有の機能用のエクスポート・ファイルです。 as400_libc.exp ファイルは、 libc.a の PASE for i 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	このファイルは、IBM i データベース機能用のエクスポート・ファイルです。 libdb400.exp ファイルは、 (DB2 for i コール・レベル・インターフェース (CLI) がサポートする) PASE for i libdb400.a ライブラリーからのエクスポートを定義します。

関連概念

46 ページの『データベース』
PASE for i は、DB2 for i コール・レベル・インターフェース (CLI) をサポートします。 AIX および IBM i 上の DB2 CLI は、それぞれ互いのまったく同じ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ます。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。

関連資料

16 ページの『システムへの IBM PASE for i プログラムのコピー』
PASE for i で実行する予定の AIX バイナリー・ファイルは、統合ファイル・システムにコピーする必要があります。

IBM i 機能にアクセスするための IBM PASE for i API

PASE for i は、 ILE コードおよびその他の IBM i 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。

関連情報

IBM PASE for i API

IBM i 環境での IBM PASE for i プログラムの使用

PASE for i プログラムは、ジョブ内で実行中の他の IBM i プログラムを呼び出すことができ、他の IBM i プログラムは PASE for i 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。

IBM PASE for i プログラムおよびプロシージャーの実行

ジョブで PASE for i プログラムを開始し、ILE プログラムから PASE for i プロシージャーを呼び出すことができます。

PASE for i プログラムは、以下のいくつかの方法で実行できます。

- IBM i ジョブ内で
- PASE for i 対話式シェル環境から

- ILE プロシージャーからの呼び出し先プログラムとして

注: PASE for i プログラムを IBM i オペレーティング・システム上で実行する場合、 PASE for i 環境変数は ILE 環境変数に依存していないことを覚えておく必要があります。 一方の環境で変数を設定しても、他方の環境には影響を与えません。

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

PASE for i には、ILE コードが PASE for i サービスに (PASE for i 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。

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

ILE コードからの PASE for i プログラム内のプロシージャーの呼び出し

PASE for i 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、 PASE for i 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 PASE for i に (対応する PASE for i 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、PASE for i プログラムの開始時に PASE for i 環境変数 PASE_THREAD_ATTACH が Y に設定された場合のみです。

PASE for i から IBM i プログラムに結果を戻す

IBM i _RETURN() 関数を使用すると、PASE for i プログラムを呼び出し、 PASE for i 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、PASE for i 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。

関連情報

IBM PASE for i ILE プロシージャー API

_RETURN()--IBM PASE for i を終了せずに戻る

QP2SHELL() を使用した IBM PASE for i プログラムの実行

任意の PASE for i コマンド行から IBM i 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、QP2SHELL または QP2SHELL2 プログラムを使用します。

これらのプログラムは、呼び出し元のジョブの中で PASE for i プログラムを実行します。プログラムでは、PASE for i プログラムの名前がパラメーターとして渡されます。

QP2SHELL() プログラムは、新しい活動化グループで PASE for i 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。

注: QP2SHELL プログラムも QP2SHELL2 プログラムも、たいていのシェルが操作の信頼性を確保するのに必要とする標準ストリーム (stdin、stdout、および stderr は、フォーク可能なファイル記述子でなければなりません) 用の特別なセットアップは行いません。したがって、シェルまたはシェル・スクリプトを実行するための追加のプログラミングと一緒に、QP2SHELL プログラムおよび QP2SHELL2 プログラムを実行する必要があります。 API プログラム QP2TERM または QSH CL コマンドを使用すれば、追加のプログラミングなしでシェル・スクリプトを実行することができます。

次の例では、IBM i 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' '/')
```

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、上のサンプルは次のような方法でコーディングする必要があります。

```
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
```

関連情報

QP2SHELL() および QP2SHELL2()--IBM PASE for i シェル・プログラムの実行

QP2TERM() を使用した IBM PASE for i プログラムの実行

この IBM i プログラムを使用して、対話式シェル環境で PASE for i プログラムを実行します。

QP2TERM() プログラムで PASE for i 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、 PASE for i 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、バッチ・ジョブでファイル stdin、stdout、および stderr の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 PASE for i プログラムへ渡す任意の引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 PASE for i 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。

関連概念

63 ページの『IBM PASE for i シェルおよびユーティリティー』
PASE for i には、3 つのシェル (Korn 、Bourne、および C shell) および PASE for i プログラムとして実行する多数のユーティリティーがあります。PASE for i 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およびデファクト・スタンダードのコマンドが多数含まれています。

関連情報

QP2TERM()--IBM PASE for i 端末セッションの実行

IBM i プログラム内からの IBM PASE for i プログラムの実行

他の ILE プロシージャー内から Qp2CallPase() および Qp2CallPase2()ILE プロシージャーを呼び出して、PASE for i プログラムを開始し、実行することができます。

PASE for i 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、引数ストリング、および環境変数を指定してください。
Qp2RunPase() API は、呼び出し元のジョブの中で PASE for i プログラムを実行します。これは PASE for i 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 PASE for i の実行方法を制御できます。

関連情報

Qp2RunPase()--IBM PASE for i プログラムの実行

例: IBM i プログラム内からの IBM PASE for i プログラムの実行:

ここの例は、PASE for i プログラムを呼び出す ILE プログラム、およびその ILE プログラムによって呼び出される PASE for i プログラムを示しています。

注: コード例を使用することにより、 79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

例 1: PASE for i プログラムを呼び出す ILE プログラム

以下の ILE プログラムは、PASE for i プログラムを呼び出します。このサンプルに続いて、このプログラムが呼び出す PASE for i コードのサンプルを示します。

```
#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 PASE
  for i 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
     i5/OS PASE program */
  char *PASE_parm = "My Parm";
  /* Argument list for i5/OS 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 i5/OS 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 i5/OS PASE */
      arg_list,       /* Arguments for i5/OS PASE program */
      NULL);          /* Environment variable list, not used in this sample */
 }
```

例 2: ILE プログラムで呼び出される PASE for i プログラム

上記の ILE プログラムにより、以下の PASE for i プログラムが呼び出されます。

```
#include <stdio.h>

/*****************************************
  Sample:
  A simple PASE for i 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 IBM i.
  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 PASE for i program %s. Parameter value is \"%s\".\n", argv[0], argv[1]);

     return 0;
}
```

IBM i プログラム内からの IBM PASE for i プロシージャーの呼び出し

他の ILE プロシージャー内から Qp2CallPase() および Qp2CallPase2() ILE プロシージャーを呼び出して、PASE for i 環境がすでに稼働しているジョブで PASE for i プログラムを実行することができます。

最初に Qp2RunPase() API が、ジョブの中で PASE for i プログラムを開始および実行します。そのジョブで PASE for i 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。

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

関連情報

Qp2CallPase()--IBM PASE for i プロシージャーの呼び出し

例 1: IBM i プログラム内からの IBM PASE for i プロシージャーの呼び出し:

この例は、PASE for i プロシージャーを呼び出す ILE プログラムを示しています。

注: コード例を使用することにより、 79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

```
#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 PASE for i program
     * /usr/lib/start32, which starts PASE for i 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 PASE for i 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 PASE for i getpid
     * function, and print the result.  Use Qp2errnop
     * to find and print the PASE for i errno if the
     * function result was -1
     */
    int rc = Qp2CallPase(getpid_pase,
                         NULL,        // no argument list
                         signature,
                         QP2_RESULT_WORD,
                         &result)
    printf("PASE for i getpid() = %i\n", result);
    if (result == -1)
        printf("IBM i errno = %i\n", *Qp2errnop());

    /*
     *  Close the Qp2dlopen instance, and then call
     *  Qp2EndPase to end PASE for i in this job
     */
```

```
    Qp2dlclose(id);
    Qp2EndPase();
    return 0;
}
```

例 2: IBM PASE for i プロシージャーの呼び出しでポインター引数を使用する IBM i ILE プログラム:

この例では、IBM i ILE プログラムは 2 つの異なる手法を使用して、呼び出す PASE for i プロシージャーでのメモリー・ストレージの割り振りと共有を行っています。

注: 以下のコード例を使用することにより、 79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

```
/* Name: ileMain.c
 *
 * Call an PASE for i procedure from ILE
 *
 * This example uses the Qp2dlopen, Qp2dlsym, and Qp2CallPase2 ILE
 * functions to call an PASE for i function passing in parameters
 *
 * Compile like so:
 *
 * CRTBNDC PGM(mylib/ilemain)
 *         SRCFILE(mylib/mysrcpf)
 *         TERASPACE(*YES *TSIFC)
 */
#include <stdio.h>
#include <stddef.h>
#include <errno.h>
#include <qp2user.h>
/* Use EBCDIC default job CCSID in Qp2dlopen and Qp2dlsym calls */
#define JOB_CCSID 0

/* start PASE for i in this process */
void startPASE(void) {
    /* start64 starts the 64 bit version of PASE for i */
    char *start64Path="/usr/lib/start64";
    char *arg_list[2];

    arg_list[0] = start64Path;
    arg_list[1] = NULL;
    Qp2RunPase(start64Path,
               NULL,
               NULL,
               0,
               819,
               (char**)&arg_list,
               NULL);

}

/* open a shared library */

QP2_ptr64_t openlib(char * libname) {
    QP2_ptr64_t id;
    int * paseErrno;

    /* Qp2dlopen dynamically loads the specified library returning an
     * id value that can be used in calls to Qp2dlsym and Qp2dlcose */
    id = Qp2dlopen(libname,
                   (QP2_RTLD_NOW |
                    QP2_RTLD_MEMBER ),
                   JOB_CCSID);
    if (id == 0) {
        printf("Qp2dlopen failed.  ILE errno=%i\n", errno);
        if ((paseErrno=Qp2errnop()) != NULL)
```

```
            printf("Qp2dlopen failed.  PASE for i errno=%i\n", *paseErrno);
        printf("Qp2dlopen failed.  Qp2dlerror = %s\n", Qp2dlerror());
    }

    return(id);
}

/* find an exported symbol */

void * findsym(const QP2_ptr64_t id, const char * functionname) {
    void * symbol;
    int * paseErrno;

    /* Qp2dlsym locates the function descriptor for the
     * specified function */
    symbol = Qp2dlsym(id, functionname, JOB_CCSID, NULL);
    if (symbol == NULL) {
        printf("Qp2dlsym failed.  ILE errno = %i\n", errno);
        if ((paseErrno=Qp2errnop()) != NULL)
            printf("Qp2dlsym failed.  PASE for i errno=%i\n", *paseErrno);
        printf("Qp2dlsym failed.  Qp2dlerror = %s\n", Qp2dlerror());
    }
    return(symbol);
}

/* call PASE for i procedure */
int callPASE(const void * functionsymbol,
             const void * arglist,
             const QP2_arg_type_t * signature,
             const QP2_result_type_t result_type,
             void * buf,
             const short buflen) {
    int * paseErrno;
    int rc;

    /* Call Qp2CallPase2 to run the unction function */
    rc = Qp2CallPase2(functionsymbol,
                      arglist,
                      signature,
                      result_type,
                      buf,
                      buflen);
    if (rc != 0) {
        printf("Qp2CallPase failed.  rc=%i, ILE errno=%i\n", rc, errno);
        if ((paseErrno=Qp2errnop()) != NULL)
            printf("Qp2CallPase failed.  PASE for i errno=%i\n", *paseErrno);
        printf("Qp2CallPase failed.  Qp2dlerror=%s\n", Qp2dlerror());
    }
}


int main(int argc, char *argv[])
{
    /* we will call a function in PASE for i named "paseFunction"
     * the prototype for the function looks like this:
     *  int paseFunction(void * input, void * output ) */

    /* "signature" is the argument signature for the PASE routine "paseFunction" */
    const QP2_arg_type_t signature[] = {QP2_ARG_PTR64, QP2_ARG_PTR64, QP2_ARG_END};

    /* "paseFunctionArglist" are the arguments for the PASE routine "paseFunction" */
    struct {
        QP2_ptr64_t inputPasePtr;
        QP2_ptr64_t outputPasePtr;
    } paseFunctionArglist;
```

```
/* "inputString" will be one of the arguments to the PASE routine
 * "paseFunction" we will call
 * This is the string "input" in ASCII */
const char inputString[] = {0x69, 0x6e, 0x70, 0x75, 0x74, 0x00};

/* "outputILEPtr" will be a pointer to storage malloc'd from PASE heap */
char * outputILEPtr;

/* "id" is the identifier for the library opened by Qp2dlopen */
QP2_ptr64_t id;

/* "paseFunction_ptr" is the pointer to the routine "paseFunction" in PASE */
void * paseFunction_ptr;

/* "inputAndResultBuffer" is the buffer of storage shared between ILE and PASE
 * by Qp2CallPase2.  This buffer contains space for the PASE function result */
struct {
    QP2_dword_t result;
    char inputValue[6];
} inputAndResultBuffer;

int rc;
int * paseErrno;

/* start PASE for i in this process */
startPASE();

id = openlib("/home/joeuser/libpasefn.a(shr64.o)");

if (id !=0) {
    /* Locate the symbol for "paseFunction" */
    paseFunction_ptr = findsym(id, "paseFunction");

    if (paseFunction_ptr != NULL) {

        /* set input arguments for the call to paseFunction() */

        /* copy the inputString into the inputAndResultBuffer */
        strcpy(inputAndResultBuffer.inputValue, inputString);

        /* by setting inputPasePtr argument to the offset of the
         * inputValue by-address argument data in the
         * inputAndResultbuffer structure and OR'ing that with
         * QP2_ARG_PTR_TOSTACK QP2CallPase2 will "fixup" the
         * actual argument pointer passed to the PASE function
         * to point to the address (plus the offset) of the
         * copy of the inputAndResultbuffer that Qp2CallPase2
         * copies to PASE for i storage */
        paseFunctionArglist.inputPasePtr =
        (QP2_ptr64_t)((offsetof(inputAndResultBuffer, inputValue))
                      | QP2_ARG_PTR_TOSTACK);

        /* allocate memory from the PASE for i heap for an output
         * argument.  Qp2malloc will also set the PASE for i address
         * of the allocated storage in the outputPasePtr
         * argument */
        outputILEPtr = Qp2malloc(10, &(paseFunctionArglist.outputPasePtr));

        /* Call the function in PASE for i */
        rc = callPASE(paseFunction_ptr,
                      &paseFunctionArglist,
                      signature,
                      QP2_RESULT_DWORD,
                      &inputAndResultBuffer,
                      sizeof(inputAndResultBuffer));
        if (rc != 0) {
```

```
                printf("output from paseFunction = >%s<\n",
                       (char*)outputILEPtr);
                printf("return code from paseFunction = %d\n",
                       (int)inputAndResultBuffer.result);
            } /* rc != 0 */
        } /* paseFunction_ptr != NULL */
    } /* id != 0 */

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

Source code for the IBM i Procedure paseFunction that is called by the ileMain.c program:

```
/* PASE for i function to be called from ILE
 *
 * Compile with something like:
 *  xlc -q64 -c -o paseFunction.o paseFunction.c
 *  ld -b64 -o shr64.o -bnoentry -bexpall -bM:SRE -lc paseFunction.o
 *  ar -X64 -r /home/joeuser/libpasefn.a shr64.o
 *
 * The ILE side of this example expects to find libpasefn.a in
 * /home/joeuser/libpasefn.a
 *
 * The compiler options -qalign=natural and -qldbl128 are
 * necessary only when interacting with IBM i ILE programs
 * to force relative 16-byte alignment of type long double
 * (used inside type ILEpointer)
 */

#include <stdlib.h>
#include <stdio.h>
int paseFunction(void * inputPtr, void * outputPtr)
{
    /* An output string to return from PASE for i to ILE *
     * this is the string "output" in EBCDIC             */
    const char outputValue[] = {0x96, 0xa4, 0xa3, 0x97, 0xa4, 0xa3, 0x00};


    printf("Entered paseFunction The input is >%s<\n",
           (char*)inputPtr);

    /* copy the output results to the outputPtr argument */
    memcpy(outputPtr, outputValue, sizeof(outputValue));


    return(52); /* return something more interesting than 0 */
}
```

例 2 の ILE の部分で使用されるさまざまな関数

- **startPASE() 関数**

  PASE for i をプロセスで使用する前に、これを開始しておく必要があります。 これは、API (たとえば QP2SHELL、QP2TERM、または Qp2RunPase) を使用して、PASE for i アプリケーションのメインエントリー・ポイントを呼び出すことにより自動的に行うことができます。

  しかし、この例は (メインエントリー・ポイントではなく) 共用ライブラリーからエクスポートされた PASE for i 関数を呼び出しているので、PASE for i を手動で開始する必要があります。これを行うに

は、/usr/lib/start32 (32 ビット・バージョンの PASE for i を開始する) および /usr/lib/start64 (64 ビット・バージョンの PASE for i を開始する) という 2 つの PASE for i 開始ユーティリティーを使用できます。

IBM i プロセスはそれぞれ 1 つの PASE for i インスタンスしか実行させておくことができないという点にご注意ください。 Qp2ptrsize() API を使用すると、PASE for i がすでに実行されているかどうかを判別することができます。

- PASE for i が現在プロセスでアクティブになっていない場合、Qp2ptrsize() は 0 を戻します。
- PASE for i が 32 ビット・モードでアクティブになっている場合、Qp2ptrsize() は 4 を戻します。
- PASE for i が 64 ビット・モードでアクティブになっている場合、Qp2ptrsize() は 8 を戻します。

- **openlib() 関数および findsym() 関数**

  これらの関数は Qp2dlopen() および Qp2dlsym() を使用して、IBM i 共用ライブラリーをオープンし、呼び出したい関数のへのポインターを取得します。これらの関数は多くのプラットフォームにおける dlopen() ルーチンおよび dlsym() ルーチンと類似しています。

- **Qp2CallPase2 呼び出しのための引数のセットアップ**

  callPASE() 関数によって Qp2CallPase2() を呼び出す前に、main() ルーチンは、ILE と PASE for i 関数の間のインターフェースを定義する以下の変数をセットアップします。

  - signature-array 変数は、PASE for i 関数の引数を定義します。 配列中のエレメントは一般に、qsysinc/h.qp2user 組み込みファイル内の #define を使用して設定されます。
  - paseFunctionArglist 構造には、PASE for i ランタイムが引数 (関数の呼び出し時に PASE for i 関数に渡される) にマップする ILE 変数が含まれています。 paseFunctionArglist のメンバーは、シグニチャー配列で宣言されている PASE for i 関数のシグニチャーと対応します。
  - inputAndResultBuffer 構造には、関数の呼び出し時に PASE for i ランタイムが ILE と PASE for i の間の一種の共用バッファーとして使用する ILE 変数が含まれています。

    構造の最初のメンバー (この例の *result*) には、PASE for i 関数からの戻り値が含まれます。 この変数は、Qp2CallPase2() API の呼び出しの 4 つ目の引数として提供される結果タイプと一致しなければなりません。この最初のエレメントの後に来るものはいずれも、関数の呼び出し時に PASE for i 環境にコピーされるストレージを表します。

    この例の inputAndResultBuffer 構造の inputValue エレメントには、PASE for i 関数の最初の引数によって指し示されるアドレス渡しの引数データが入ります。
  - この例では、呼び出されている PASE for i 関数のポインター引数の、2 つの異なる設定方法が使用されています。
    - 関数の 2 番目の引数 paseFunctionArglist.outputPasePtr は、Qp2malloc() 関数を呼び出すことによって設定されます。 Qp2malloc() は、PASE for i ランタイム・ヒープからメモリーを割り振り、割り振られたストレージに、ILE ポインターと PASE for i ポインターの両方を戻します。
    - 最初の引数 paseFunctionArglist.inputPasePtr は、inputAndResultBuffer 構造の inputValue エレメントのオフセットに設定されます (OR によって qp2user.h #define QP2_ARG_PTR_TOSTACK とつながれます)。

      これは、PASE for i ランタイムに、アドレス (inputAndResultBuffer.inputValue が PASE for i メモリーにコピーされた) による PASE for i 関数の呼び出しによって提供される実際のポインター値を変更するように命令します。

- **callPASE() 関数**

この関数は、Qp2CallPase2() API、および main() ルーチンで設定される引数を使用して、PASE for i 関数を呼び出します。

- **プロセスでの PASE for i の終了**

  PASE for i 関数を呼び出した後、PASE for i 共有ライブラリーをアンロードするために Qp2dlclose() API が呼び出され、この例の最初に呼び出された start64 プログラムを終了するために Qp2EndPase() が呼び出されます。

Java からの IBM PASE for i ネイティブ・メソッドの使用

PASE for i 環境で実行される PASE for i ネイティブ・メソッドを、 Java プログラムから使用することができます。

PASE for i ネイティブ・メソッドのサポートには、PASE for i ネイティブ・メソッドのすべてのネイティブの IBM i Java ネイティブ・インターフェース (JNI) を使用する機能と、ネイティブ IBM i Java 仮想マシン (JVM) から PASE for i ネイティブ・メソッドを呼び出す機能が含まれます。

関連情報

IBM IBM PASE for i native methods for Java

環境変数の処理

PASE for i 環境変数は ILE 環境変数に依存しません。一方の環境で変数を設定しても、他方の環境には影響を与えません。

ただし、PASE for i プログラムの実行に使用する方法に応じて、 ILE から PASE for i に変数をコピーすることができます。

対話式 PASE for i 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、 PASE for i に渡されます。 PASE for i 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 PASE for i セッションの環境変数

PASE for i 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。 PASE for i 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 PASE for i 環境変数をコピーすることができます。これは、PASE for i `system` ユーティリティーのデフォルトの動作でもあります。

関連情報

QP2SHELL() および QP2SHELL2()--IBM PASE for i シェル・プログラムの実行

QP2TERM()--IBM PASE for i 端末セッションの実行

systemCL()--IBM PASE for i 用の CL コマンドの実行

IBM PASE for i 環境変数

IBM PASE for i プログラムからの IBM i プログラムおよびプロシージャーの呼び出し

PASE for i では、ILE プロシージャー、Java プログラム、OPM プログラム、IBM i API、および IBM i 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。

IBM i プログラムおよびプロシージャーの一般構成要件

PASE for i プログラム環境から IBM i 環境に呼び出しを行う場合、一般に、呼び出される側のプログラムは必ず活動化グループに *CALLER を指定してコンパイルされる必要があります。それには、以下の理由があります。

- PASE for i (Qp2RunPase API によって呼び出される) を開始した活動化グループ内で実行するコードだけが、Qp2CallPase などの ILE API を使用して PASE for i プログラムと対話できる。
- ILE ランタイムは、マルチスレッド・ジョブ内の活動化グループを破壊する必要がある場合、 (PASE for i `fork` が作成するすべてのジョブはマルチスレッド対応)、ジョブ全体を終了してしまう場合がある (PASE for i 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 ランタイム機能を使用して 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ことに伴う問題を避けることができます。

関連タスク

19 ページの『IBM i 機能を使用するための IBM PASE for i 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 IBM i 共用ライブラリーでは直接サポートされていない PASE for i 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。

ILE プロシージャーの呼び出し

以下のステップを行って、ILE プロシージャーを準備し、PASE for i プログラムから呼び出すことができます。

PASE for i プログラムから ILE プロシージャーを呼び出す際には、まず ILE プロシージャーを、テラスペース用に使用可能化し、テキストを適切な CCSID に変換し、変数および構造をセットアップして準備する必要があります。

1. **テラスペース用に ILE プロシージャーを使用可能にする**

   PASE for i 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。 ILE モジュールがそのようにコンパイルされていないと、PASE for i 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 の記憶モデルが正しくありません。) が記録されます。
2. **テキストを適切な CCSID に変換する**

ILE と PASE for i 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。

3. **変数と構造をセットアップする**

   PASE for i 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、PASE for i 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、 i5/OS システム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、 PASE for i と ILE の間で渡される引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数、または 8 バイト以下の浮動小数点引数が含まれる場合、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 への標準 C リンクで、1 バイトまたは 2 バイトの整数引数を 4 バイトに、 4 バイトの浮動小数点引数を 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。
   - **引数リスト:** 引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() 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数リストを動的に作成することができます。

PASE for i 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. PASE for i 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、_ILELOADX() API を使用します。

   PASE for i 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、_ILESYMX() 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、_ILESYMX() API を使用します。
3. ILE プロシージャーを呼び出して、 PASE for i 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() または _ILECALLX() API を使用します。

関連資料

20 ページの『ヘッダー・ファイルのコピー』
この情報を参考にして、IBM i を実行しているシステムから AIX を実行しているシステムにヘッダー・ファイルをコピーします。

関連情報

size_ILEarglist()--IBM PASE for i() 用の ILE 引数リスト・サイズの計算

build_ILEarglist()--IBM PASE for i 用の ILE 引数リストの作成

_ILELOADX()--IBM PASE for i 用の ILE 結合プログラムのロード

_ILESYMX()--IBM PASE for i 用のエクスポート済み ILE シンボルの検索

_ILECALLX()--IBM PASE for i 用の ILE プロシージャーの呼び出し

 ILE 概念 PDF

例: ILE プロシージャーの呼び出し:

このコード例は、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 PASE for i 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。

以下のコード例には、2 つのプロシージャーがあります。 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、PASE for i システムが提供するメソッドを使用した、 _ILECALL() 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、手動による引数リストの作成を示しています。

注: コード例を使用することにより、 79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

例 1: PASE for i 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
 * ILEpointer to a procedure that is exported by that
 * service program.
 */

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

   /* _ILELOADX() loads the service program */
   actmark = _ILELOADX("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. _ILESYMX()
    * extracts an ILE procedure pointer from the
    * service program activation
    */

   ILEtarget = (ILEpointer*)ROUND_QUAD(ILEtarget_buf);
   rc = _ILESYMX(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_t result_type = RESULT_INT32;
static arg_type_t signature[] =
{
    ARG_INT32,
    ARG_MEMPTR,
    ARG_FLOAT64,
    ARG_UINT8,      /* requires #pragma nowiden in ILE code */
    sizeof(aggregate),
    ARG_INT16,
    ARG_END
};

/*
 * simple_wrapper 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 best_wrapper.
 */
int simple_wrapper(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
     * 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 ILEpointer
     * 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;

/*
 * best_wrapper 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 best_wrapper(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 ILEpointer
     * 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);
        }
      }
}
```

例 2: ILE C コード

ここでは、IBM i 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 i5/OS 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;
}
```

例 3: プログラムを作成するためのコンパイラー・コマンド

PASE for i 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 IBM i 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、PASE for i 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)
```

IBM PASE for i からの IBM i プログラムの呼び出し

IBM i アプリケーションを作成する際に、既存の PASE for i プログラム (*PGM オブジェクト) を利用することができます。さらに、systemCL() 機能を使用して CL CALL コマンドを実行することができます。

IBM i プログラム内から PASE for i 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。
このメソッドの処理速度は systemCL() ランタイム機能より高速ですが、 (PGMCALL_ASCII_STRINGS を指定しない限り) 文字ストリング引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。

関連タスク

44 ページの『IBM PASE for i からの IBM i コマンドの実行』
IBM i 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、PASE for i プログラムの機能を拡張することができます。

関連情報

_PGMCALL()--IBM PASE for i 用の IBM i プログラムの呼び出し

例: IBM PASE for i からの IBM i プログラムの呼び出し:

以下の例は、_PGMCALL ランタイム機能を使用して、PASE for i プログラム内のプログラムを呼び出す方法を示しています。

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。

注: コード例を使用することにより、 79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

```
/* This example uses the IBM i PASE _PGMCALL function to call the IBM i
API QSZRTVPR. The QSZRTVPR API is used to retrieve information about
i5/OS 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[])
{

   /* IBM i API's (including QSZRTVPR) typically expect character
      parameters to be in EBCDIC.  However, character constants in
      PASE for i 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 i5/OS 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 IBM i 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 IBM i 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 IBM i QSZRTVPR API from PASE for i */
   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("IBM i Option 33 is NOT installed\n");
   else
      printf("IBM i Option 33 IS installed\n");

   return(0);
}
```

IBM PASE for i からの IBM i コマンドの実行

IBM i 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、PASE for i 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、PASE for i プログラム内から IBM i コマンドを実行します。

IBM i コマンドを PASE for i から実行すると、systemCL ランタイム機能が文字ストリング引数の ASCII から EBCDIC への変換を自動的に処理し、別のジョブのプログラムを呼び出せるようにします。

関連タスク

42 ページの『IBM PASE for i からの IBM i プログラムの呼び出し』
IBM i アプリケーションを作成する際に、既存の PASE for i プログラム (*PGM オブジェクト) を利用することができます。さらに、systemCL() 機能を使用して CL CALL コマンドを実行することができます。

関連情報

systemCL()--IBM PASE for i 用の CL コマンドの実行

例: IBM PASE for i からの IBM i コマンドの実行:

以下の例は、PASE for i プログラムで CL コマンドを実行する方法を示しています。

注: コード例を使用することにより、79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

以下の例は、PASE for i 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);
   }
}
```

IBM PASE for i プログラムと IBM i の相互作用

PASE for i の機能を使用するように IBM i 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。

関連タスク

19 ページの『IBM i 機能を使用するための IBM PASE for i 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 IBM i 共用ライブラリーでは直接サポートされていない PASE for i 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。

通信

PASE for i は一般に、ソケット通信において AIX および Linux との互換性があります。

PASE for i 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、詳細な点では他のオペレーティング・システム (Linux など) と異なる場合があります。

PASE for i 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、 PASE for i 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) IBM i 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。

IBM i 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、PASE for i 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。

関連情報

ソケット・プログラミング

バークレー・ソフトウェア・ディストリビューションとの互換性

UNIX 98 互換性

データベース

PASE for i は、DB2 for i コール・レベル・インターフェース (CLI) をサポートします。 AIX および IBM i 上の DB2 CLI は、それぞれ互いのまったく同じ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ます。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、 PASE for i 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` は IBM i バージョンのものを使っ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、 PASE for i では実行できません。

IBM i 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、IBM i データベース (DB2 for i) と PASE for i 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 PASE for i をインプリメントする際に、 PASE for i システム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 から拡張 2 進化 10 進交換コード (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 PASE for i 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、PASE for i CCSID を明示的に指定する必要があります。

PASE for i 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 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 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、 PASE for i CCSID および PASE for i 環境変数 `LANG` を、ジョブの言語および CCSID 属性の PASE for i に相当する最適なものを使用して設定します。

libc.a の拡張によって、PASE for i 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、PASE for i 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: オーバーライドを有効にするには、CCSID オーバーライド関数 SQLOverrideCCSID400() 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。

PASE for i プログラムでの DB2 for i CLI の使用

PASE for i 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、PASE for i 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。 PASE for i CLI 関数は多くの場合、対応する ILE CLI 関数を呼び出して、必要な操作を実行します。

注: DB2 CLI を PASE for i プログラムで使用する場合、以下の点を考慮してください。

- SQLGetSubString は CLOB/DBCLOB フィールドをサブストリング化する場合、常に EBCDIC ストリングを戻す。 SQLGetSubString は LOB データ・タイプに対してのみ使用されます。
- 結果セット (テーブル・タイプ) の列 4 である SQLTables は常に EBCDIC として戻される。
- PASE for i プログラムで図形文字データを扱うには、そのデータはプログラム内で `wchar` として入力されている必要があります。これによってデータベースは図形文字のみの 2 バイト文字を Unicode/UCS-2 に変換します。そうしないと、データベースはデータの CCSID と IBM i ジョブの CCSID の間で変換を行います。データベースは、EBCDIC 図形文字と CCSID との間の変換 (Qp2RunPase() API または SQLOverrideCCSID400() API のいずれかによる) をサポートしません。

関連資料

20 ページの『ヘッダー・ファイルのコピー』
この情報を参考にして、IBM i を実行しているシステムから AIX を実行しているシステムにヘッダー・ファイルをコピーします。

21 ページの『エクスポート・ファイルのコピー』
この情報を参考にして、IBM i を実行しているシステムから 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。

関連情報

QP2TERM()--IBM PASE for i 端末セッションの実行

QP2SHELL() および QP2SHELL2()--IBM PASE for i シェル・プログラムの実行

_SETCCSID()--IBM PASE for i CCSID の設定

SQLOverrideCCSID400()--IBM PASE for i 用の SQL CLI CCSID のオーバーライド

SQL CLI

例: PASE for i プログラムでの DB2 for i CLI 関数の呼び出し:

この例は、DB2 for i SQL コール・レベル・インターフェースを使用して DB2 for i にアクセスする PASE for i プログラムを示しています。

注: コード例を使用することにより、 79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

```
/* i5/OS PASE DB2 for i5/OS example program
 *
 * To show an example of an PASE for i program that accesses
 * DB2 for i via SQL CLI
 *
 * Program accesses System i Access database, 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

AIX や Linux など、ほとんどのオペレーティング・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 IBM i 機能では、EBCDIC 文字エンコードが使用されます。

いくつかの IBM i オブジェクト・タイプに対してコード化文字セット ID (CCSID) 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指定することができます。

PASE for i バイト・ストリーム・ファイルには、CCSID 属性があります。 PASE for i 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。 IBM i 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)、 PASE for i 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 PASE for i 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。

PASE for i 共用ライブラリーに付属する AIX API を使用する場合、 PASE for i はほとんどのデータ変換を処理します。 PASE for i プログラムは、PASE for i 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、 PASE for i 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) IBM i 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。

関連概念

『ファイル・システム』
PASE for i 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。

ファイル・システム

PASE for i 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

外部装置との間でやり取りされる入出力は、IBM i オペレーティング・システム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。これとは逆に、AIX や Linux などのオペレーティング・システムは通常、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。 IBM i オペレーティング・システムでは、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

PASE for i 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、IBM i オペレーティング・システムによって使用されるコード化スキームに自動的に変換されますが、オープン・ファイルで読み取りまたは書き込みされるデータは変換されません。

ファイル記述子の使用

PASE for i 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、PASE for i 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。

PASE for i および ILE C は、標準入出力 (stdin、stdout、および stderr) について同じストリームを使用します。 PASE for i 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。ただし、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、PASE for i は PASE for i 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、PASE for i 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 PASE for i 拡張版である F_MAP_XPFFD を使用して、 PASE for i 記述子を ILE 番号に割り当てることができます。これは、 PASE for i によって作成されなかった ILE 記述子のファイル操作を PASE for i 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 IBM i システム固有の拡張版である STX_XPFFD_PASE を使用すると、 PASE for i プログラムは PASE for i 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 PASE for i 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、 PASE for i 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、 PASE for i 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 PASE for i コードまたは ILE C コードがファイル記述子 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。

PASE for i ランタイムは一般に、PASE for i 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、 (PASE for i CCSID とジョブ・デフォルト CCSID の間で) 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 for i システム固有の環境変数は、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。

PASE for i 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。

関連概念

54 ページの『データ・エンコード』
AIX や Linux など、ほとんどのオペレーティング・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 IBM i 機能では、EBCDIC 文字エンコードが使用されます。

関連情報

統合ファイル・システム

グローバリゼーション

PASE for i は AIX のランタイムをベースにしているため、 PASE for i 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。

PASE for i 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。

PASE for i 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。 PASE for i 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: PASE for i 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

PASE for i 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 PASE for i 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

PASE for i 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、PASE for i 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数が、 PASE for i 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。

PASE for i ランタイムは、 PASE for i 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 PASE for i 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 PASE for i には、UTF-8 ロケールを使用してください。 IBM i オペレーティング・システムには、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、PASE for i CCSID を IBM-euc*XX* コード・セットのエンコード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・データが他のコード化スキーム (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。

PASE for i 変換オブジェクトとロケールの保管場所

PASE for i の変換オブジェクトとロケールは、 IBM i 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、PASE for i がインストールされるときに、インストールされる IBM i 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 PASE for i 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 PASE for i ランタイムは、ASCII (または UTF-8) で動作します。

関連タスク

6 ページの『IBM PASE for i のインストール』
PASE for i は、オプションでインストールできるオペレーティング・システム・コンポーネントです。これを使用したり、PASE for i サポートを必要とするソフトウェアを実行するには、PASE for i をインストールする必要があります。

関連情報

IBM i グローバリゼーション

IBM PASE for i ロケール

_SETCCSID()--IBM PASE for i CCSID の設定

メッセージ・サービス

PASE for i 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 PASE for i シグナルを POST するには、PASE for i Qp2SignalPase() API を使用できます。QP2SHELL() プログラムと PASE for i fork() 関数は、すべての ILE シグナルを対応する PASE for i 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 IBM i 例外メッセージを、自動的に、対応する PASE for i シグナルに変換します。このようにして、PASE for i アプリケーションは、システムによって変換された IBM i シグナルを処理することによって、すべての PASE for i 例外を処理することができます。

PASE for i には、 IBM i のメッセージ処理を直接制御できるようにする、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

IBM i メッセージ・サポート

IBM i 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、ジョブが実行されたりコンパイルされたりしたときに IBM i オペレーティング・システムやアプリケーションで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」表示画面が表示されたら、F10 (コマンド入力画面からの詳細メッセージの組み込み) を押し、続いて Shift + F6 を押します。 「すべてのメッセージの表示」画面が表示されて、最新のメッセージが示されます。特定のメッセージの詳細情報を表示するには、そのメッセージの上にカーソルを移動してから、F1 (ヘルプ) を押します。

アクティブ・ジョブ (WRKACTJOB) コマンドの処理

IBM i オペレーティング・システム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。

関連情報

Qp2SignalPase()--IBM PASE for i シグナルの POST

IBM PASE for i プログラムが使用するためのランタイム機能

アクティブ・ジョブ (WRKACTJOB) コマンドの処理

実行管理機能

IBM PASE for i シグナルの処理

IBM PASE for i アプリケーションからの印刷出力

PASE for i 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された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
```

関連情報

Rfile - レコード・ファイルの読み書きをする

疑似端末 (PTY)

PASE for i は、AT&T 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 PASE for i 上で機能します。

PASE for i 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。

PASE for i での 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 PASE for i 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-style スタイルのデバイスの場合、PASE for i 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-style スタイルのデバイスの場合は、PASE for i 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 従属デバイスと BSD 基本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 疑似端末 (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。

  注: コード例を使用することにより、 79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

```
#!/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
```

PTY デバイスの詳細については、 AIX リソース Web ページ (英語) 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、 PASE for i プログラムは、IBM i オペレーティング・システム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。

PASE for i オペレーティング・システムで IBM i 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・ファイル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。

PASE for i 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、IBM i 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。 IBM i オペレーティング・システムにおいて、 *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 PASE for i 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						

PASE for i のユーザー・プロファイル

IBM i オペレーティング・システムでは、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、 PASE for i 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 Portable Operation System Interface X (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。

IBM i オペレーティング・システムには、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、IBM i オペレーティング・システムには、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。 たとえば、このような特権の 1 つである *ALLOBJ は、ファイル・アクセスに対する任意アクセス制御を指定変更します。これは、AIX や Linux などのオペレーティング・システムでのルート特権の一般的な使用法です。

root アクセスを使用する移植アプリケーションでは、*ALLOBJ 権限の付与の対象となりう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するのが、セキュリティーという点ではより望ましい措置といえます。このようにすれば、QSECOFR を使用しなくて済みます。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。 AIX または Linux などのオペレーティング・システムとは異なり、IBM i オペレーティング・システムでは、ユーザーのグループ・メンバーシップは必要ありません。 IBM i オペレーティング・システムでは、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。

IBM i 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。

PASE for i 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。 PASE for i 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、アドレス・スペースに入ると、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。

PASE for i 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。 PASE for i プログラムに対するシステム呼び出しは、IBM i オペレーティング・システムによって処理されます。 このインターフェースでは、PASE for i 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。

関連情報

セキュリティー

実行管理機能

IBM i オペレーティング・システムは、システム上の他のジョブを処理するのと同じ方法で PASE for i プログラムを処理します。

関連情報

実行管理機能

IBM PASE for i プログラムのデバッグ

PASE for i 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・ファイルのライブラリー・サポートを提供しています。加えて、たとえば診断メッセージ用のジョブ・ログや、システム・オペレーター・メッセージ・キュー QSYSOPR へ送信される重大メッセージなどの、IBM i オペレーティング・システムの既存の機能を使用することもできます。

アプリケーションによっては、PASE for i 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。

1. アプリケーションが IBM i の統合 (たとえば、 DB2 for i または ILE 機能との統合) 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。
2. その後、PASE for i dbxと IBM i デバッグ機能 (ジョブ・ログなど) を組み合わせて使用して、IBM i オペレーティング・システム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。

PASE for i での dbx の使用

PASE for i 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)。 PASE for i 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 PASE for i 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、IBM AIX オペレーティング・システム: ライブラリー Web サイト (英語) を参照してください。 dbx コマンド行で help と入力することもできます。

IBM i デバッグ・ツールの使用

IBM i アプリケーションをデバッグするために、 PASE for i では以下のツールを使用することができます。

- System i5® Debugger は、PASE for i アプリケーションのデバッグ用の特定のサポートを提供します。
- 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。

関連情報

System i デバッガー

WebSphere Development Studio: ILE C/C++ プログラマーの手引き PDF

IBM AIX オペレーティング・システム: リソース (英語)

パフォーマンスの最適化

最大限のパフォーマンスを得るには、ローカル・ストリーム・ファイル・システムにアプリケーションのバイナリー・ファイルを保管します。

バイナリー・ファイル (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、 PASE for i 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 PASE for i 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低下します。これは、それぞれの PASE for i fork() 操作ごとに新しい IBM i 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。

関連情報

パフォーマンス

IBM PASE for i シェルおよびユーティリティー

PASE for i には、3 つのシェル (Korn 、Bourne、および C shell) および PASE for i プログラムとして実行する多数のユーティリティーがあります。PASE for i 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およびデファクト・スタンダードのコマンドが多数含まれています。

PASE for i のデフォルト・シェル (/QOpenSys/usr/bin/sh) は、Korn シェルです。

PASE for i シェルおよびユーティリティーにアクセスするために、Run および i5/OS 端末セッション (QP2TERM) プログラムを呼び出すことができます。このプログラムは、PASE for i コマンドの入力先のコマンド行を備えた対話式表示を提示します。シェルまたはユーティリティーも含め、どの PASE for i プログラムを実行するにも、Run および任意の PASE for i プログラム (QP2SHELL) API を呼び出すことができます。

たいていの PASE for i ユーティリティーに、ディレクトリー /usr/bin 内の QShell と同じ名前が付いている (オプションと動作も似通っています) ので、PASE for i ユーティリティーは、ディレクトリー /QOpenSys/usr/bin または /QOpenSys/usr/sbin 内に用意されています。 PASE for i シェルの実行時には、一般的に、PASE for i PATH 環境変数には、ディレクトリー /QOpenSys/usr/bin、/QOpenSys/usr/bin/X11、および /QOpenSys/usr/sbin が組み込まれます。 PASE for i 環境変数の初期値の設定の詳細は、任意の PASE for i プログラム (QP2SHELL) API の実行を参照してください。

関連概念

1 ページの『IBM i 7.1 の新機能』
PASE for i のトピック集の中で、新規または大幅に変更された情報を以下に記載しています。

3 ページの『IBM PASE for i の概念』
PASE for i は、IBM i オペレーティング・システム上で稼働する AIX アプリケーション用の統合ランタイム環境です。

関連タスク

24 ページの『QP2TERM() を使用した IBM PASE for i プログラムの実行』
この IBM i プログラムを使用して、対話式シェル環境で PASE for i プログラムを実行します。

関連資料

10 ページの『AIX ソースのコンパイル』
IBM i でのインストールをサポートする AIX コンパイラー製品のいずれかをインストールして、PASE for i 環境でプログラムをコンパイルすることができます。

関連情報

IBM PASE for i 端末セッションの実行

IBM PASE for i コマンド

PASE for i 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、同じ動作をします。しかし、PASE for i コマンドは、いくつかの点で AIX コマンドとは異なります。

以下のリストは、PASE for i コマンドと AIX コマンドの相違を説明しています。

- 表示操作用と UNIXジョブ制御用のたいていの PASE for i コマンドは、aixterm や xterm コマンドで開始されるセッションのような、テレタイプライター (TTY) セッションでのみ稼働します。このような機能は、5250 ワークステーション・デバイス (QP2TERM プログラムで示される画面も含む) では稼働しません。
- 一般的に、PASE for i は、システム管理用の AIX 上に装備されているインターフェースはサポートしません。 たとえば、PASE for i は、AIX System Management Interface Tool (SMIT) や、SMIT データベースを必要とする機能をサポートしません。
- 基本的に、IBM i オペレーティング・システムは EBCDIC システムです。 PASE for i のシェルおよびユーティリティーは ASCII で稼働し、一般的に、ストリーム・データの自動変換は行いません。 ASCII と EBCDIC の間の変換を行うためのツール (たとえば、iconv() 関数) が必要になるかもしれません。

  QShell インタープリターおよびユーティリティーとは異なり、たいていの PASE for i シェルおよびユーティリティーは、ストリーム・ファイル・データのコード化文字セット ID (CCSID) の自動変換は行いません。ただし、PASE for i ユーティリティー・システムと、QShell コマンドを実行するすべての PASE for i ユーティリティーは例外です。その理由は、CL コマンドや QShell コマンドが標準入力から読み取ったり、標準出力やエラーに書き込んだりするデータの CCSID 変換サポートが提供されるからです。

  QShell Java ユーティリティー (たとえば、Java コマンド) を実行する PASE for i ユーティリティーは、PASE for i CCSID に一致するように Java file.encoding プロパティーを設定します。それにより、Java プログラムによって読み取り/書き込みされるストリーム・データの、PASE for i CCSID への変換やこの CCSID からの変換が行われます。特定の file.encoding 値が強制的に使用されるようにするには、このユーティリティーの実行の前に、PASE for i 環境変数 *PASE_JAVA_ENCODING* を設定します。

- たいていのシステム・リソースの場合、IBM i オペレーティング・システムは、大/小文字の区別のない名前を使用します。ただし、AIX では、そのようなシステム・リソースでも、大/小文字の区別のある名前が付いています。たとえば、ユーザー名およびグループ名や、ルート・ファイル・システム内のオブジェクト名がそれに該当します。 PASE for i のシェルおよびユーティリティー機能によっては、IBM i で大/小文字の区別のない名前が付けられているリソースに対しても、大/小文字文字の一致を義務付けている場合や、AIX であれば通常は小文字となる名前が大文字で返される場合があります。たとえば、PASE for i シェルのファイル名の拡張子では大/小文字が区別されるので、/QSYS.LIB ファイル・システムの総称名に一致させるには、大文字を指定する必要があります。

```
          ls /qsys.lib/qgpl.lib/GEN*.PGM
rather than
          ls /qsys.lib/qgpl.lib/gen*.pgm
```

- 大/小文字の区別を実施して、ILE サポートで使用されるディレクトリーおよびファイルの名前との衝突が起きないようにするため、PASE for i の大半のディレクトリーおよびファイル (シェルおよびユーティリティーも含む) は、/QOpenSys ファイル・システムに保管されます。特に、PASE for i のシェルおよびユーティリティーは、/QOpenSys/usr/bin および /QOpenSys/usr/sbin (AIX 上の /usr/bin および /usr/sbin ではなく) に保管されます。

以下の PASE for i コマンドに加えて、各 PASE for i 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。各 PASE for i シェルでサポートされる組み込みコマンドの詳細と、次の PASE for i コマンドの大部分の詳細については、 AIX リソース Web ページ (英語) を参照してください。

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A	
admin	ソース・コード制御システム (SCCS) ファイルを作成して制御します。
aixterm	Enhanced X Window System 端末エミュレーターを初期化します。
alias	別名を定義または表示します。
appletviewer	Web ブラウザーなしで、QShell appletviewer コマンドを実行して、Java アプレットを実行します。
apply	一連のパラメーターに対してコマンドを適用します。
apt	QShell apt コマンド (Java アノテーション処理ツール) を実行します。
ar	リンケージ・エディターで使用される索引付きライブラリーを保守します。
as	アセンブラーを実行します。
attr	QShell attr コマンドを実行して、統合ファイル・システムのオブジェクト属性を表示または変更します。
awk	パターンに一致する行をファイル内で見つけ出してから、その行に対して指定されたアクションを実行します。
B	
banner	大きな文字の ASCII 文字列を標準出力に書き出します
basename	ストリング・パラメーターの基本ファイル名を戻します。
bc	任意の精度の算術計算言語用のインタープリターを提供します。

bdiff	diff コマンドを使用して、非常に大きなファイルでの相違点を見つけ出します。
bfs	ファイルをスキャンします。
bg	ジョブをバックグラウンドで実行します。
bsh	Bourne シェルを呼び出します。
C	
cat	ファイルの連結または表示を行います。
cd	現行ディレクトリーを変更します。
cdc	SCCS 差分内のコメントを変更します。
chgrp	ファイルまたはディレクトリーのグループ所有権を変更します。
chmod	許可モードを変更します。
chown	ファイルに関連付けられているユーザーを変更します。
chroot	コマンドのルート・ディレクトリーを変更します。
cksum	ファイルのチェックサムおよびバイト・カウントを表示します。
clear	C 端末画面をクリアします。
clrtmp	QShell clrtmp コマンドを実行して、ディレクトリー /tmp をクリアします。
cmp	2 つのファイルを比較します。
colrm	ファイルから列を抽出します。
comb	SCCS 差分を結合します。
comm	ソート済みの 2 つのファイルに共通する行を選択またはリジェクトします。
command	シンプルなコマンドを実行します。
compress	データを圧縮します。
cp	ファイルをコピーします。
cpio	アーカイブ・ストレージとディレクトリーとの間で、ファイルのコピーを行います
csh	C シェルを呼び出します。
csplit	コンテキストに従ってファイルを分割します。
cut	ファイルの各行から選択されたバイト、文字、フィールドを書き出します
D	
date	日付または時刻を表示または設定します。
dbx	PASE for i プログラムをデバッグして実行する環境を提供します。
dc	任意の精度の整数算術計算用の対話式卓上計算機を提供します。
dd	ファイルを変換してコピーします。
df	ファイルシステム上のスペースに関する情報を報告します
diff	テキスト・ファイルを比較します。
diff3	3 つのファイルを比較します。

dircmp	2 つのディレクトリーとその中にあるファイルの内容を比較します
dirname	指定されたパスの最後の部分を除いて、標準出力にすべてを書き込みます。
dspcat	メッセージ・カタログの全部または一部を表示します。
dspmsg	メッセージ・カタログから選択されたメッセージを表示します。
du	ディスクの使用法を要約します。
dump	オブジェクト・ファイルの選択された部分をダンプします。
E	
echo	標準出力に文字列を書き出します。
ed	行単位でテキストを編集します。
edit	新規ユーザー用のシンプルな行エディターを提供します。
egrep	ファイルの内容をパターンで検索します
env	現在の環境を表示し、コマンド実行用の環境を設定します
ex	画面表示上で対話式で行を編集します。
execerror	エラー・メッセージを標準エラーに書き出します。
expand	タブをスペースに変更したうえで、標準出力に書き出します。
expr	引数を式と評価します。
extcheck	QShell extcheck コマンドを実行して、Java アーカイブの競合を検出します。
F	
false	ゼロ以外の終了値 (false) を戻します。
fc	コマンド・ヒストリー・リストを処理します。
fg	ジョブをフォアグラウンドで実行します。
fgrep	ビルド・プロセスでサポートされているフォーマットで、図リストを生成します。
file	ファイル・タイプを判別します。
find	一致する式を持ったファイルを見つけ出します。
fold	有限幅の出力デバイスに合わせて、長い行を折り返します。
G	
gencat	メッセージ・カタログを作成および変更します。
get	指定されたバージョンの SCCS ファイルを作成します。
getconf	システム構成変数値を標準出力に書き込みます。
getjobid	QShell getjobid コマンドを実行して、プロセス ID の IBM i ジョブ名を判別します。
getopt	コマンド行のフラグおよびパラメーターを解析します。
getopts	コマンド行引数を処理し、有効なオプションかどうかを検査します。
grep	ファイルの内容をパターンで検索します

H	
hash	コマンド・パス名を記憶またはレポートします。
head	1 つ以上のファイルの最初の数行または、最初の数バイトを表示します
hostname	現在のホスト・システムの名前を設定または表示します。
I	
iconv	ある 1 つのコード・ページ・コード化スキームから別のコード化スキームへ、文字のエンコード方式を変換します。
id	指定されたユーザーのシステム識別を表示します
idlj	QShell idlj コマンドを実行して、IDL から Java へのコンパイラーを実行します。
indent	C 言語プログラムの再フォーマット設定を行います。
install	コマンドをインストールします。
ipcrm	QShell ipcrm コマンドを実行して、プロセス間通信オブジェクトを除去します。
ipcs	QShell ipcs コマンドを実行して、プロセス間通信オブジェクトを表示します。
J	
jar	QShell jar コマンドを実行して、Java ファイルをアーカイブします。
jarsigner	QShell jarsigner コマンドを実行して、Java アーカイブのシグニチャーを署名または検証します。
java	QShell java コマンドを実行して、Java インタープリターを実行します。
javac	QShell javac コマンドを実行して、Java プログラムをコンパイルします。
javadoc	QShell javadoc コマンドを実行して、Java 文書を生成します。
javah	QShell javah コマンドを実行して、Java クラスの C ヘッダーまたはスタブ・ファイルを生成します。
javakey	QShell javakey コマンドを実行して、Java セキュリティー・キーを管理します。
javap	QShell javap コマンドを実行して、コンパイル済みの Java プログラムを逆アセンブルします。
jobs	現行セッション中のジョブの状況を表示します。
join	2 つのファイルのデータ・フィールドを結合します。
K	
keytool	QShell keytool コマンドを実行して、Java のキーおよび証明書を管理します。
kill	実行中のプロセスにシグナルを送信します
ksh	Korn シェルを呼び出します。
ksh93	拡張 Korn シェルを呼び出します。
L	

ld	オブジェクト・ファイルをリンクします。
ldedit	XCOFF 実行可能ファイル・ヘッダーを変更します。
lex	入力ストリームの単純な字句解析でパターンを突き合わせる C または C++ 言語プログラムを生成します。
line	標準入力から 1 行読み取ります。
ln	ファイルをリンクします。
locale	現在のロケールまたはすべてのパブリック・ロケールに関する情報を書き出します。
logger	システム・ログ中に項目を作成します。
logname	ログイン名を表示します
look	ソート済みファイル内で行を見つけ出します。
lorder	オブジェクト・ライブラリー内のメンバー・ファイルの最適な順序を見つけ出します。
ls	ディレクトリーの内容を表示します
M	
m4	ファイルを前処理し、マクロ定義を拡張します。
make	プログラム・グループの保守、更新、および再生成を行います。
makekey	暗号鍵を生成します。
mkcatdefs	メッセージ・ソース・ファイルをプリプロセスします。
mkdir	1 つ以上の新規ディレクトリーを作成します
mkfifo	先入れ先出し (FIFO) 特殊ファイルを作成します。
mkfontdir	フォント・ファイルのディレクトリーから fonts.dir ファイルを作成します。
mknod	特殊ファイルを作成します。
more	ファイルの内容を一度に 1 つの画面に表示します。
mv	ファイルを移動します。
mwm	AIXwindows ウィンドウ・マネージャー (MWM) を実行します。
N	
native2ascii	QShell native2ascii コマンドを実行して、PASE for i CCSID でエンコードされている文字を Unicode エンコードに変換します。
nawk	新バージョンの awk を呼び出します。
newform	テキスト・ファイルのフォーマットを変更します。
nice	コマンドを低い優先順位あるいは高い優先順位で実行します
nl	ファイルの行に番号を付けます。
nm	オブジェクト・ファイルのシンボル・テーブルを表示します。
nohup	ハングアップなしでコマンドを実行します。
O	
od	指定されたフォーマットでファイルを表示します。

orbd	QShell orbd コマンドを実行して、Java オブジェクト・リクエスト・ブローカー・デーモンを実行します。
P	
pack	ファイルを圧縮します。
pack200	Java アーカイブ圧縮ツールの QShell pack200 コマンドを実行
pagesize	システムのページ・サイズを表示します。
paste	複数のファイルの行、または 1 つのファイルの連続する行をマージします
patch	ファイルに対して変更を適用します。
pax	アーカイブ・ファイルのメンバーを抽出し、書き出し、そしてリストします。ファイルとディレクトリーの階層をコピーします。
pcat	ファイルを解凍し、それを標準出力に書き込みます。
pg	ディスプレイに合わせてファイルをフォーマット設定します。
policytool	QShell policytool コマンドを実行して、Java ポリシー・ファイルを作成して管理します。
pr	ファイルを標準出力へ書き出します。
printenv	環境変数の値を表示します
printf	フォーマットされた出力を書き出します。
prs	ソース・コード制御システム (SCCS) ファイルを表示します。
ps	プロセスの現在の状況を示します。
psh	POSIX (Korn) シェルを呼び出します。
pwd	作業ディレクトリーのパス名を表示します
Q	
qsh	QShell コマンドを実行します。
qsh_inout	QShell コマンドを実行します。
qsh_out	QShell コマンドを実行します。
R	
ranlib	アーカイブ・ライブラリーをランダム・ライブラリーに変換します。
read	標準入力から 1 行読み取ります。
red	行単位でテキストを編集します。
regcmp	パターンを C 言語の CHAR 型宣言にコンパイルします。
reset	端末を初期化します。
resize	TERMCAP 環境変数および端末の設定値を、現行ウィンドウ・サイズに設定します。
rev	ファイルの各行内の文字を反転します。
Rfile	QShell Rfile コマンドを実行して、IBM i レコード・ファイルの読み取りまたは書き込みを行います。

rgb	カラー用の X Window システム・サーバーで使用されるデータベースを作成します。
rm	ファイルまたはディレクトリーを除去 (リンク解除) します。
rmdel	SCCS ファイルから差分を除去します。
rmdir	ディレクトリーを除去します
rmic	QShell rmic コマンドを実行して、Java RMI スタブをコンパイルします。
rmid	QShell rmid コマンドを実行して、Java RMI 活動化システムを実行します。
rmiregistry	QShell rmiregistry コマンドを実行して、Java リモート・オブジェクト・レジストリーを開始します。
rtl_enable	共用オブジェクトを再リンクして、実行時リンカーでそのオブジェクトを使用できるようにします。
runcat	出力データを mkcatdefs コマンドから gencat コマンドにパイピングします。
S	
sact	現在の SCCS のファイル編集状況を表示します。
serialver	QShell serialver コマンドを実行して、Java クラスのバージョン番号を戻します。
sccs	SCCS コマンドの管理プログラム。
sccsdiff	2 つのバージョンの SCCS ファイルを比較します。
sdiff	2 つのファイルを比較し、相違点を横並びのフォーマットで表示します。
sed	ストリーム・エディターを提供します。
servertool	QShell servertool コマンドを実行して、Java IDL サーバー・ツールを実行します。
setccsid	QShell setccsid コマンドを実行して、統合ファイル・システム・オブジェクトの CCSID を設定します。
setmaps	端末マップまたはコード・セット・マップを設定します。
sh	デフォルト (Korn) シェルを呼び出します。
size	Extended Common Object File Format (XCOFF) オブジェクト・ファイルのセクション・サイズを表示します。
sleep	あるインターバルの間、実行を一時中断します。
slibclean	カーネルおよびライブラリー・メモリー内の現在未使用のモジュールを除去します。
snapcore	コア・ファイルの情報を収集します。
sort	ファイルのソート、既にソートされているファイルのマージ、およびソート済みかどうかの判別のためのファイルの検査を行います。
split	ファイルをいくつかの断片に分割します。
strings	オブジェクトまたはバイナリー・ファイル内で、印刷可能ストリングを検索します。

strip	バインダーおよびシンボリック・デバッグ・プログラムで使用される情報を除去して、XCOFF オブジェクト・ファイルのサイズを削減します。
stty	ワークステーションの操作パラメーターを設定し、リセットし、そしてレポートします。
sum	ファイルのチェックサムおよびブロック数を表示します。
syslogd	システム・メッセージのログ記録をとります。
システム	CL コマンドを実行します。
sysval	QShell sysval コマンドを実行して、IBM i のシステム値またはネットワーク属性を表示します。
T	
tab	スペースをタブに変更します。
tabs	端末上でタブ停止位置を設定します。
tail	指定された位置から、ファイルを標準出力に書き出します
tar	アーカイブを操作します。
tee	プログラムの出力を表示して、その出力をファイルにコピーします
test	条件式を評価します。
tic	terminfo 記述ファイルをソース・フォーマットからコンパイル済みフォーマットに変換します。
time	コマンドの実行時刻を印刷します。
tnameserv	QShell tnameserv コマンドを実行して、Java 命名サービスにアクセスできるようにします。
touch	ファイルのアクセス時刻と変更時刻を更新します。
tput	端末依存情報を terminfo データベースに照会します。
tr	文字を変換します。
trace	選択されたシステム・イベントを記録します。
trbsd	文字 (BSD バージョン) を変換します。
trcoff	トレース・データの収集を停止します。
trcon	トレース・データの収集を開始します。
trcstop	トレース機能を停止します。
true	ゼロの終了値 (true) を戻します。
tset	端末を初期化します。
tsort	順番に並べられたペア (位相ソート) の順不同リストをソートします。
tty	端末の絶対パス名を標準出力に書き出します。
type	コマンド・タイプの説明を書き出します。
U	
ulimit	ユーザーのリソース制限を設定またはレポートします。
umask	ファイル・モード作成マスクを表示または設定します。
unalias	別名定義を除去します。
uname	現在のオペレーティング・システムの名前を表示します
uncompress	圧縮ファイルを復元します。

unexpand	タブを復元したうえで、標準出力に書き出します。
unget	前の SCCS get コマンドをキャンセルします。
unifdef	ファイルから ifdef 行を除去します。
uniq	ファイルの反復行を削除します。
unpack	ファイルを拡張表示します。
unpack200	Java アーカイブ解凍ツールの QShell unpack200 コマンドを実行します。
untab	タブをスペースに変更します。
V	
val	SCCS ファイルを検証します。
vc	割り当て済みの値を ID キーワードに置き換えます。
vedit	全画面表示でファイルを編集します。
vi	全画面表示でファイルを編集します。
view	読み取り専用モードで vi エディターを開始します。
W	
wait	プロセス ID が終了するまで待機します。
wc	ファイル内の行数、ワード数、およびバイト数または文字数を数えます
what	ファイル内の識別情報を表示します。
which	別名およびパスも含め、プログラム・ファイルを見つけ出します (csh (C shell) コマンドのみ)。
X	
X	X サーバーを実行します。 PASE for i は、仮想フレーム・バッファー処理のみをサポートします。
xargs	パラメーター・リストを構成し、コマンドを実行します。
xauth	X サーバーへの接続で使用される許可情報を編集して表示します。
xhost	現在のプライマリー・システムで誰が Enhanced X Window System にアクセスできるかを制御します。
xlsfonts	X Window System のフォント・リストを表示します。
xmodmap	X サーバーでキー・マップを変更します。
xset	X Window System 環境のオプションを設定します。
xterm	X Window System の端末エミュレーターを提供します。
xwd	Enhanced X Window System ウィンドウのイメージをダンプします。
xwud	Enhanced X Window System ウィンドウのダンプ・イメージを取り出して表示します。
Y	
yacc	コンテキストのない文法仕様から成る入力から、LALR(1) 解析プログラムを生成します。
yes	肯定応答の出力を反復して作成します。
Z	
zcat	圧縮ファイルを標準出力に拡張表示します。

関連情報

Java プログラムの開発ユーティリティー

IBM PASE for i system ユーティリティー

PASE for i system ユーティリティーは、CL コマンドを実行します。デフォルトでは、このコマンドで作成されたすべてのスプール出力は、標準出力に書き出されます。このコマンドから送信されたすべてのメッセージは、標準出力または標準エラーに書き出されます (CL コマンドが例外メッセージを送信するかどうかに応じて)。

想定外の結果を招かないためには、ILE 環境変数 *QIBM_USE_DESCRIPTOR_STDIO* を Y または I に設定し、それによって PASE for i のランタイムおよび ILE の C ランタイムが記述子標準入力および出力を使用するようにします。QP2TERM プログラムが IBM i シェルおよびユーティリティーの実行で使用する PASE for i ジョブでは、この変数はデフォルトで Y または I に設定されています。

Syntax

system [-beEhiIkKnOpqsv] *CL-command* [*CL-parameters* ...]

オプション

-b CL コマンドで使用される標準ストリームを、強制的にバイナリー・モードにします。

このオプションを省略すると、system コマンドは、CL コマンドが標準入力から読み取ったすべてのデータを、PASE for i CCSID からジョブのデフォルトの CCSID に変換します。 system コマンドは、標準出力または標準エラーに書き込まれたデータを、ジョブのデフォルト CCSID から PASE for i CCSID に変換します。このオプションは、オプション -E、-I、または -O を除き、すべての標準ストリームが CCSID 変換されないようにします。

-e CL コマンドの実行の前に、PASE for i 環境変数を ILE 環境変数にコピーします。

このオプションを省略すると、どの ILE 環境変数も設定されません。さらに、ILE 環境から変数が脱落する場合や、PASE for i 環境のもとのとは異なる変数値を持つことになる場合があります。

たいていの変数の場合、コピーにはオリジナルと同じ名前が付きますが、一部の環境変数の ILE コピーの名前には、システムによって接頭部 PASE_ が付け加えられます。デフォルトでは、PASE for i 環境変数 *SHELL*、*PATH*、*NLSPATH*、および *LANG* のコピー時に、システムによって接頭部が付け加えられます。名前の接頭部が付け加えられる変数を制御するには、変数名をコロンで区切ったリストを PASE for i 環境変数 *PASE_ENVIRON_CONFLICT* 内に保管しておきます。

名前に接頭部 ILE_ の付いた PASE for i 環境変数はすべて、ILE 環境に 2 回コピーされます。一方のコピーは同じ変数名を使用し、もう一方のコピーは、接頭部なしの名前を使用します。たとえば、PASE for i 環境に、*ILE_PATH* という名前の変数が入っている場合、この変数の値が、ILE 環境において *ILE_PATH* および *PATH* の両方の変数を設定するときに使用されます。

-E CL コマンドで使用される標準エラー・ストリームの CCSID 変換を強制的に行います。

このオプションを指定すると、system コマンドは、CL コマンドが標準エラーに書き込んだすべてのデータを、ジョブのデフォルトの CCSID から PASE for i CCSID に変換します。このオプションは、標準エラー・ストリームのオプション -b を指定変更します。

-h system コマンドで使用できる構文の要旨を標準出力に書き出します。

-i system ユーティリティーが稼働しているのと同じプロセス (IBM i ジョブ) 内で、CL コマンドを実行します。

オプション -i を省略すると、ILE spawn APIを使って作成された別のプロセスで CL コマンドが実行されます。 ILE 環境変数 *QIBM_MULTI_THREADED* を Y に設定していない限り、この別のプロセスは、マルチスレッド対応にはなりません。たいていの CL コマンドは、マルチスレッド・ジョブではサポートされていません。

-I CL コマンドで使用される標準入力ストリームの CCSID 変換を強制的に行います。

このオプションを指定すると、system コマンドは、CL コマンドが標準入力から読み取ったすべてのデータを、PASE for i CCSID からジョブのデフォルトの CCSID に変換します。このオプションは、標準入力ストリームのオプション -b を指定変更します。 CL コマンドが標準入力を読み取る場合の標準入力に対してのみ、CCSID 変換を使用する必要があります。その理由は、system コマンドが実行する処理では、すべての標準入力データの読み取りと変換が、CL データがそのデータを使用するかどうかに関係なく試みられるため、標準入力ストリームは、CL コマンドが読み取る範囲外にはみ出す可能性があるからです。

-k CL コマンドで生成されたすべてのスプール・ファイルを保持します。

このオプションを省略すると、スプール出力ファイルは、その内容がテキスト行として標準出力に書き出された後で削除されます。オプション -s を使用すると、オプション -i には何の効果もなくなります。

-K CL コマンドを実行する場所である IBM i ジョブのジョブ・ログを強制的にとります。

このオプションを省略すると、想定外のエラーの発生時のみジョブ・ログを作成することができます。

-n CL コマンドから送信されるメッセージの、標準出力または標準エラーに書き出されるどのテキスト行にも、IBM i メッセージ ID を付けません。

このオプションを省略すると、IBM i の事前定義メッセージから書き出されるテキスト行のフォーマットは *XXX1234: message text* となります。ただし、*XXX1234* は、IBM i メッセージ ID です。 -n はメッセージ ID を抑止するので、*message text* のみがストリームに書き込まれます。オプション -q を使用すると、オプション -n には何の効果もなくなります。

-O CL コマンドで使用される標準出力ストリームの CCSID 変換を強制的に行います。

このオプションを指定すると、system コマンドは、CL コマンドが標準出力に書き込んだすべてのデータを、ジョブのデフォルトの CCSID から PASE for i CCSID に変換します。このオプションは、標準出力ストリームのオプション -b を指定変更します。

-p このオプションは無視されます。

PASE for i system ユーティリティーは常に、CL コマンドを実行するプログラムへ送信されたメッセージのみを処理します (これは、QShell system ユーティリティーがオプション -p を処理するやり方です)。

-q CL コマンドから送信された IBM i メッセージのどのテキスト行も標準出力または標準エラーに書き出しません。

このオプションを省略すると、CL コマンドから送信されたメッセージは、受信されると、ジョブのデフォルト CCSID から PASE for i CCSID に変換されてから、CL コマンドが例外メッセージを送信するかどうかに応じて、標準出力または標準エラーにテキスト行として書き出されます。

-s CL コマンドで生成されたスプール出力ファイルを処理しません。

このオプションを省略すると、CL コマンドで生成されたスプール出力は、ジョブのデフォルトの CCSID から PASE for i CCSID に変換されてから、標準出力に書き出されます。その後、スプール出力ファイルは削除されます。

-v CL コマンドの実行の前に、完全な CL コマンド・ストリングを標準出力に書き出します。

オペランド

CL-command と任意の *CL-parameters* オペランドを、シングル・スペースをはさんで連結すると、CL コマンド・ストリングが形成されます。 CL コマンドとパラメーター値を引用符で囲んで、PASE for i シェルによって特殊文字 (括弧やアスタリスクなど) が拡張されないようにする必要があります。

CL コマンドのパラメーター値に引用符が必要な場合 (小文字または組み込みブランクを使ったテキスト・パラメーターなど)、その引用符を、引用符付きストリングの内側に指定する必要があります。その理由は、PASE for i system ユーティリティーに渡されたすべての引数から外側の引用符が、PASE for i シェルによって除去されるからです。

終了状況

CL コマンド分析プログラムまたはコマンド処理プログラムから例外メッセージが送信された場合、system ユーティリティーは 255 の終了状況を戻します。エラー・メッセージは常に、コマンドを実行する IBM i ジョブのジョブ・ログに表示されます。また、オプション -q を指定しない限り、標準出力または標準エラーにも送信されます。

CL コマンドの処理から例外メッセージが送信されなかった場合、system ユーティリティーは、CL コマンドが呼び出したプログラムで設定された終了状況を戻します。ただし、そのプログラムが終了状況を設定していない場合は、ゼロを戻します。

例

以下の例は、同じパラメーター値を使って CRTDTAARA CL コマンドを実行するいくつかの方法を示しています。オプション -bOE は、標準出力および標準エラー (標準入力は対象外) の CCSID 変換を強制的に行います。 *char パラメーター値を引用符で囲んで、一連のファイル名としてその値が PASE for i シェルによって拡張されないようにする必要があります。 TEXT パラメーターは、小文字および組み込みブランクが入っているので、2 セットの引用符で囲む必要があります。

```
      system -bOE "crtdtaara mydata *char text('Output queue text')"
または
      system -bOE crtdtaara mydata "*char text('Output queue text')"
または
      system -BOE crtdtaara mydata '*char' "text('Output queue text')"
```

以下の例は、system ユーティリティーが CALL CL コマンドを実行して、2 つのパラメーターを受け入れるプログラムを呼び出す方法を示しています。オプション -i は、CL コマンドを実行する追加のプロセスの作成というオーバーヘッドを回避します。他のオプションは何も指定していないので、標準入力、標準出力、および標準エラーの CCSID 変換が行われます。呼び出されたプログラムは、CL 規則にのっとって、最初のパラメーターは大文字 (ARG1) に変換されていて、2 番目のパラメーターは変更されていない (arg2) とみなします。

```
      system -i "call mypgm (arg1 'arg2')"
```

関連概念

『IBM PASE for i qsh、qsh_inout、および qsh_out コマンド』
PASE for i qsh、qsh_inout、および qsh_out コマンドは、QShell コマンドを実行します。これらのコマンドは、PASE for i system コマンドを使用して、PASE for i 環境変数を ILE 環境にコピーしてから、ディレクトリー /usr/bin 内のリンクを通して QShellコマンド・プログラムを呼び出します。

IBM PASE for i qsh、qsh_inout、および qsh_out コマンド

PASE for i qsh、qsh_inout、および qsh_out コマンドは、QShell コマンドを実行します。これらのコマンドは、PASE for i system コマンドを使用して、PASE for i 環境変数を ILE 環境にコピーしてから、ディレクトリー /usr/bin 内のリンクを通して QShellコマンド・プログラムを呼び出します。

PASE for i qsh、qsh_inout、および qsh_out コマンドはいずれも、QShell qsh コマンドの構文と動作を備えています。ただし、ASCII と EBCDIC の間の、標準入力および出力のエンコード方式変換のサポートが追加されています。 PASE for i system コマンドは、エンコード方式の変換サポートを備えています。
PASE for i qsh、qsh_inout、または qsh_out (ディレクトリー /QOpenSys/usr/bin 内の) にリンクする他のすべてのコマンド名は、リンクと同じベース名の付いたディレクトリー /usr/bin 内の QShell コマンドと同じ構文と動作を提供します。

qsh コマンドと qsh_inout コマンドは、標準入力、標準出力、および標準エラーの ASCII と EBCDIC の間のエンコード方式変換を実行します。 qsh_out コマンドは、標準出力と標準エラーのエンコード方式変換のみを実行します。

想定外の結果を招かないためには、ILE 環境変数 *QIBM_USE_DESCRIPTOR_STDIO* を Y または I に設定し、それによって PASE for i のランタイムおよび ILE の C ランタイムが記述子標準入力および出力を使用するようにします。 QP2TERM プログラムが IBM i シェルおよびユーティリティーの実行で使用する PASE for i ジョブでは、この変数はデフォルトで Y または I に設定されています。

Syntax

```
qsh [command-options]

qsh_inout [command-options]

qsh_out [command-options]
```

例

QShell コマンドが標準入力からの読み取りを行わない場合、qsh_out コマンドを使用して (qsh または qsh_inout コマンドではなく)、入力ストリームが不用意に位置変更されないようにする必要があります。以下の例は、qsh_out コマンドを使用して、read コマンドで処理されるストリームが位置変更されないようにする一方で、ファイル myinput の内容を単に標準出力にエコーするだけです。

```
while read ; do
    qsh_out -c "echo $REPLY"
done < myinput
```

以下の例は QShell cat コマンドを使用して、IBM i ソース・データベース・ファイルのテキストを (ASCII) PASE for i CCSID に変換し、その結果を ascii_sqlcli.h という名前のストリーム・ファイルに保管します。この場合、PASE for i cat コマンドの使用時には追加されない行終了文字をストリームに挿入するために、QShell ユーティリティー・サポートが使用されています。

```
qsh_out -c 'cat /qsys.lib/qsysinc.lib/h.file/sqlcli.mbr' > ascii_sqlcli.h
```

シンボリック・リンク /QOpenSys/usr/bin/getjobid -> qsh_out を使用して QShell getjobid コマンドを実行する PASE for i getjobid コマンドが、システムから提供されます。以下の例は、IBM i シェルを実行している PASE for i ジョブの名前を判別するために QShell ユーティリティーを実行する 2 種類の方法を示しています。最初の方法のほうが、QShell インタープリターを実行しなくて済むので、効率はより高くなります。 PASE for i シェルは、変数 *$$* をシェルのプロセス ID に拡張します。 QShell getjobid コマンドは、行を標準出力に書き出します。

```
getjobid $$
```

```
qsh_out -c "/usr/bin/getjobid $$"
```

関連資料

74 ページの『IBM PASE for i system ユーティリティー』
PASE for i system ユーティリティーは、CL コマンドを実行します。デフォルトでは、このコマンドで作成されたすべてのスプール出力は、標準出力に書き出されます。このコマンドから送信されたすべてのメッセージは、標準出力または標準エラーに書き出されます (CL コマンドが例外メッセージを送信するかどうかに応じて)。

関連情報

qsh - QShell コマンド言語インタープリター

例: IBM PASE for i

以下の例が PASE for i 情報として提供されています。

注: コード例を使用することにより、 79 ページの『コードに関するライセンス情報および特記事項』 の条件に同意することになります。

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

PASE for i プログラムからの PASE for i プログラムの呼び出し

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

PASE for i プログラムでの DB2 for i 関数の使用

- PASE for i プログラムでの DB2 for i CLI 関数の呼び出し

IBM PASE for i の関連情報

IBM Redbooks 資料、Web サイト、およびその他の Information Center トピック集には、PASE for i のトピック集に関連した情報が揃っています。以下の PDF ファイルのいずれも表示または印刷できます。

IBM Redbooks

Bringing PHP to Your iSeries® Server (英語) : この資料で説明しているステップバイステップのインプリメンテーションには、PASE for i で稼働する CGI 版の Hypertext Preprocessor (PHP) が必要です。

Web サイト

- ロードマップおよびリソースの使用可能化 (英語)  (http://www.ibm.com/servers/enable/site/porting/index.html)

  この Web サイトでは、PASE for i と、ご使用のシステムにアプリケーションを移植するための他のソリューションとを比較しています。

- IBM AIX オペレーティング・システム: リソース (英語)  (http://www.ibm.com/systems/power/software/aix/resources.html)

  この Web サイトは、AIX のコマンドおよびユーティリティーに関する情報を提供しています。

その他の情報

- PASE for i API

  PASE for i API の以下の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
  - PASE for i プログラムが使用するためのランタイム機能

  PASE for i 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、PASE for i 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。

- PASE for i ランタイム・ライブラリー

  PASE for i 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。PASE for i 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。PASE for i 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。

関連資料

2 ページの『IBM PASE for i の PDF ファイル』
PASE for i の情報の PDF ファイルを表示または印刷できます。

コードに関するライセンス情報および特記事項

IBM は、お客様に、すべてのプログラム・コードのサンプルを使用することができる非独占的な著作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。

強行法規で除外を禁止されている場合を除き、IBM、そのプログラム開発者、および供給者は「プログラム」および「プログラム」に対する技術的サポートがある場合にはその技術的サポートについて、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。

いかなる場合においても、IBM および IBM のサプライヤーならびに IBM ビジネス・パートナーは、その予見の有無を問わず発生した以下のものについて賠償責任を負いません。

1. データの喪失、または損傷。

2. 直接損害、特別損害、付随的損害、間接損害、または経済上の結果的損害
3. 逸失した利益、ビジネス上の収益、あるいは節約すべかりし費用

国または地域によっては、法律の強行規定により、上記の責任の制限が適用されない場合があります。

付録. 特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒242-8502
神奈川県大和市下鶴間1623番14号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、IBM 機械コードのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。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。従って 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。サンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. _年を入れる_.

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

プログラミング・インターフェース情報

本書「IBM PASE for i」には、プログラムを作成するユーザーが IBM i のサービスを使用するためのプログラミング・インターフェースが記述されています。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc.の米国およびその他の国における商標です。

使用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21